Diss. ETH No. 20115

Optimization of Railway Operations

Algorithms, Complexity, and Models

A dissertation submitted to

ETH Zurich

for the degree of Doctor of Sciences

presented by

Holger-Frederik Robert Flier
Dipl. Wirt.-Inf. (University of Cologne)
born November 11, 1979,
citizen of Grevenbroich, Germany

accepted on the recommendation of

Prof. Dr. Peter Widmayer, ETH Zurich
examiner

Prof. Dr. Anita Schöbel, University of Göttingen
co-examiner

Dr. Matúš Mihalák, ETH Zurich
co-examiner

2011

Abstract

In this thesis we investigate a broad range of algorithmic problems in both freight and passenger railway transportation. These problems have in common that one seeks to optimize railway operations using a given infrastructure. In several interesting problem settings we investigate how the current planning practice, which is mainly based on manual decisions and experience, can be supported and improved. To that end, we create a mathematical model for each problem, analyze its complexity, and seek to develop practically efficient algorithms. In many cases, we were able to obtain real-world data on which we evaluate our methods experimentally. The topics covered are:

Utilizing Delay Data We show how to utilize historic delay data of trains in the following two application settings. The first application deals with adding a new train to an existing timetable. Here, planners would like to avoid an increase in overall delay once the additional train is in operation. We propose to predict a certain measure for the risk of delay of the additional train using linear regression models on the basis of historic delay data. We show how to integrate these models into a combinatorial shortest path model in order to compute a set of Pareto optimal train schedules with respect to risk of delay and travel time.

The second application is about finding delay dependencies in the data. A delayed train may cause some of the delay of another train, so called secondary delay. In order to improve the punctuality of future timetables, planners would like to know about such dependencies. We present efficient algorithms to detect dependencies due to resource conflicts and due to timetabled connections.

Optimizing Operations at Classification Yards In single wagon load traffic, freight trains may consist of cars of different customers and have various destinations. In order to route each car to its destination, trains are disassembled into single cars which are then regrouped to form new outbound trains. This process is performed at so called classification yards, which are the bottleneck of single wagon load traffic. One problem in this process is how to allocate the various tracks of a classification yard to outbound trains such that every outbound train can be formed on a separate classification track and

depart on time. We study the complexity of several variants of the problem and show a relation to special list coloring problems in interval graphs. Based on these results, we devise heuristics as well as and an integer programming formulation for the problem. As a case study, we consider a real-world problem instance from the Hallsberg classification yard in Sweden.

Theoretical Models for Dispatching Due to delays that occur during daily operations, railway timetables are never executed as planned. The task of a dispatcher is to return to the planned timetable as closely as possible by deciding, e.g., about breaking or maintaining connections, rerouting and rescheduling of trains, or even reassigning crews and rolling stock. Here, we study theoretical problems that are motivated by the dispatching process and are of general interest in theoretical computer science.

In particular, we study variants of the vertex disjoint paths problem in planar graphs where paths have to be selected from given sets of paths. We investigate the problem as a decision, maximization, and routing-in-rounds problem. Although all considered variants are NP-hard in planar graphs, restrictions on the location of the terminals allow for polynomial time algorithms or approximation algorithms.

A related interesting open problem in graph theory is that of finding a maximum independent set in outerstring graphs. We present a polynomial-time algorithm for the subclass of outersegment graphs where every segment is either horizontally or vertically aligned.

Finally, we consider the reassignment of crews in the presence of delays. We address a theoretical abstraction of the problem of making optimal crew swap decisions during operations. We provide efficient algorithms for the local case and show that optimizing crew swaps over the whole railway network is NP-hard.

Zusammenfassung

In dieser Arbeit betrachten wir verschiedenste algorithmische Probleme im Zugverkehr, sowohl im Personen- als auch im Güterverkehr. Diesen Probleme ist das Ziel gemeinsam, den täglichen Betrieb auf einer gegebenen Infrastruktur zu optimieren. Wir untersuchen anhand mehrerer interessanter Probleme, wie der gegenwärtige Planungsablauf, der hauptsächlich auf manuellen Entscheidungen und Erfahrung basiert, unterstützt und verbessert werden kann. Dazu erstellen wir für jedes Problem ein mathematisches Modell, analysieren dessen Komplexität und entwickeln, nach Möglichkeit, praktisch effiziente Algorithmen. In vielen Fällen konnten wir reale Daten erhalten, auf denen wir unsere Methoden experimentell evaluieren. Die Themen lauten im Einzelnen:

Einsatz von Verspätungsdaten Wir zeigen anhand von zwei Anwendungen, wie man von Verspätungsdaten von Zügen profitieren kann. In der ersten Anwendung soll ein zusätzlicher Zug zu einem vorhandenen Fahrplan hinzugefügt werden. Hierbei möchten man eine Zunahme an Verspätungen durch den zusätzlichen Zug vermeiden. Wir schlagen vor, ein gewisses Risikomass für die Verspätung des zusätzlichen Zuges mittels linearer Regressionsmodelle auf der Basis der Verspätungsdaten zu berechnen. Wir zeigen, wie man diese Modelle in ein kombinatorisches Kürzeste-Wege-Modell integriert um eine Menge von Pareto-effizienten Trassen in Bezug auf Verspätungsrisiko und Reisezeit zu berechnen.

In der zweiten Anwendung geht es darum, Abhängigkeiten zwischen Verspätungen in den Daten zu finden. Ein verspäteter Zug kann eine sogenannte Folgeverspätung eines anderen Zuges verursachen. Um zukünftige Fahrpläne zu verbessern, ist man an der Erkennung solcher Abhängigkeiten interessiert. Wir stellen effiziente Algorithmen vor, die Abhängigkeiten entdecken, wenn sie auf gemeinsam genutzter Infrastruktur oder auf geplanten Zugverbindungen beruhen.

Optimierung des Betriebs auf Rangierbahnhöfen Im Einzelwagenverkehr kann ein Frachtzug aus Wagons verschiedener Kunden und unterschiedlicher Zielorte zusammengesetzt sein. Um jeden Wagon an seinen Zielort transportieren zu können, werden die Züge in

einzelne Wagons zerlegt und zu neuen Zügen zusammengesetzt. Dieser Prozess geschieht in Rangierbahnhöfen, die einen Engpass im Einzelwagenverkehr darstellen. Ein Problem innerhalb dieses Prozesses ist die Zuordnung von Rangiergleisen zu Frachtzügen, so dass jeder abfahrende Zug auf einem separaten Gleis zusammengestellt werden und pünktlich abfahren kann. Wir untersuchen die Komplexität verschiedener Problemvarianten und zeigen die Verbindung zu speziellen Listen-Färbungsproblemen in Intervallgraphen auf. Ferner entwerfen wir sowohl Heuristiken als auch ein ganzzahliges mathematisches Programm für das Problem. Als Fallstudie betrachten wir eine Instanz des Rangierbahnhofs Hallsberg in Schweden.

Theoretische Modelle zur Dispositionsplanung Wegen Verspätungen im täglichen Betrieb werden Eisenbahnfahrpläne nie ausgeführt wie geplant. Die Aufgabe der Dispositionsplanung ist es, Entscheidungen zu treffen, um schnellstmöglich zum geplanten Ablauf zurückzukehren, z.B. durch Halten oder Brechen einer Verbindung, ändern von Fahrstrassen und Abfahrtzeiten, oder sogar die Neuzuweisung von Personal und Rollmaterial. In diesem Teil untersuchen wir theoretische Probleme, welche durch die Disposition motiviert sind und von generellem Interesse in der Theoretischen Informatik sind.

Im Einzelnen untersuchen wir Varianten des Problems, knotendisjunkte Pfade in planeren Graphen zu finden, wobei die Pfade aus einer gegebenen Menge gewählt werden müssen. Wir betrachten das Problem als Entscheidungs-, Maximierungs- und Färbungsproblem. Obwohl alle Varianten NP-schwer in planeren Graphen sind, gibt es effiziente Algorithmen für Spezialfälle hinsichtlich der Platzierung Endknoten der Pfade.

Ein verwandtes offenes Problem in der Graphentheorie ist das Problem, eine maximale unabhängige Menge in Outerstring Graphen zu finden. Wir stellen einen effizienten Algorithmus für die Unterklasse von Outersegment Graphen vor, wobei jedes Segment entweder horizontal oder vertikal ausgerichtet ist.

Schliesslich betrachten wir die Neuzuweisung von Besatzungen bei Verspätungen. Wir behandeln eine theoretische Abstraktion des Problems, optimale Neuzuweisungen im laufenden Betrieb zu finden. Wir stellen effiziente Algorithmen für den lokalen Fall vor und zeigen, dass die Optimierung von Neuzuweisungen über das gesamte Netz NP-schwer ist.

Acknowledgements

I would like to thank my supervisor Peter Widmayer for the opportunity to pursue my PhD studies at ETH, which I consider a great privilege. I am very grateful for his cordiality, encouragement, and trust.

I would also like to thank Anita Schöbel for our interesting discussions and her spontaneous willingness to be co-examiner of my thesis.

A big thank you goes to my co-supervisors Marc Nunkesser and Matúš Mihalák for their strong commitment to our projects and for the many good times we had together, both in and outside of research.

Further, I would like to thank all my co-authors: Anna Zych, Jens Maue, Markus Bohlin, and last but not least Rati Gelashvili. A special thank you goes to Thomas Graffagnino from SBB for our pleasant cooperation (including a trip inside the driver's cabin!).

Coming to work has always been a great pleasure. I would like to thank all members of our research group and CADMO for the friendly atmosphere.

Finally, I thank my parents for their love and support. And Donja, for everything.

Contents

Chapter 1

Introduction

Among railway passengers across Europe an invariant seems to hold, namely that everybody thinks railways could do better in terms of punctuality. Some blame the railway companies for poor planning. Others might wonder whether it is the complexity of railway systems that is an obstacle to improvement which is hard to overcome. Compared to the airline industry, many railway operators are indeed far behind in applying planning tools that make use of mathematical optimization. However, the problem instances in railways are much larger and harder to solve than in the airline industry, suggesting that advances both in algorithms and computer hardware were not ripe enough for practical applicability in the past. For example, even at major airports, once a plane has taken off, there is usually little interdependence with other planes, such that a delay at take-off can often be compensated during flight. By contrast, many trains at a railway station compete for the same tracks or are part of a timetabled connection. Because of such interdependencies, the delay of one train may propagate to many others. Even worse, the amount of delay that can be compensated between stops is rather limited, especially if trains of different speeds share the same track infrastructure.

Despite this complexity, there are recent success stories demonstrating the practical applicability of algorithms, mathematics, and operations research to railway optimization. The first known computer generated timetable is the 2005 timetable of the subway of Berlin, which was made possible by the application of novel col-

umn generation techniques [64]. In December 2006, a new railway timetable was introduced for the Netherlands. Unlike previous timetables, which are obtained from the past one by smaller manual modifications, the new timetable was constructed from scratch using sophisticated operations research techniques [62]. In both cases, railway operations have been improved significantly for the passengers. These examples show that railway companies should embrace innovations from computer science, mathematics, and operations research in order to better utilize existing infrastructure.

This thesis addresses several problems that arise in various areas and stages of railway planning. The problems we consider range from applied problems, which we encountered in cooperation with railway operators, to purely theoretical problems, where we investigate interesting fundamental problems underlying real-world applications. We are particularly interested in designing practically efficient algorithms or to prove that no polynomial time algorithm exists for the problem at hand, assuming that $P \neq NP$. For computationally hard problems, we also seek to develop approximation algorithms, heuristics, and suitable mathematical programming formulations. Where possible, we seek to evaluate our methods by computational experiments on real-world data. In the following, we give a brief summary of each of the topics of this thesis.

Utilizing Delay Data. The first part of this thesis deals with practical problems in which we make extensive use of historic delay data of trains. The projects in this part were carried out in cooperation with the Swiss Federal Railways, who also provided us with their data. These data contain a wealth of information that can be exploited for planning purposes.

Scheduling Additional Trains on Corridors. First, we consider a common practical planning problem of offering an additional train service to accommodate temporary traffic, such as, e.g., during a major sports event. When adding a new train to an existing timetable, planners have to take the overall expected risk of delay in the new timetable into account. Typically, this can be a very laborious task involving a lot of manual planning and detailed simulations. To support the planners, we propose to predict the risk of delay of an additional train using linear regression models on the basis of the delay data. We show how to integrate these models into a combinatorial shortest

path model in order to compute a set of Pareto optimal train schedules with respect to risk and travel time. We discuss the consequences of choosing different types of linear regression models and notions of risk with respect to the algorithmic complexity of the resulting combinatorial problems. We conduct computational experiments in order to demonstrate the quality of these models.

Mining Railway Delay Dependencies. The propagation of delays between trains has a considerable impact on railway operations. Ideally, planners would like to create timetables that avoid such propagation as much as possible. To improve existing timetables, planners would like to be able to identify systematic delay dependencies between trains, i.e., pairs of trains where the delay of one train is mostly caused by the delay of the other. We present efficient algorithms to detect two of the most important types of dependencies, namely dependencies due to conflicts on tracks and due to timetabled connections. We give experimental results that demonstrate the practical applicability of our algorithms.

Optimizing Operations at Classification Yards. The second part of this thesis deals with the operation of classification yards (also called marshalling, shunting, or hump yards). At their core, planning problems at classification yards have a wealth of interesting combinatorial problems, many of which have not been studied before. Further, classification yards constitute a bottleneck of rail-freight operations, in particular in single wagon load traffic (where freight trains may consist of cars of various customers and destinations), which is in direct competition to transportation by truck. Transportation by truck has higher external costs to society than railways in terms of fatal accidents, emissions, and abrasion of roads, see, e.g., [33]. We seek to improve planning at classification yards in order to make rail-freight traffic more competitive.

Track Allocation at Classification Yards. We consider the everyday process of forming outbound trains from cars of inbound trains at rail-freight classification yards. Given the arrival and departure times as well as the composition of the trains, we study the problem of allocating the various tracks of a classification yard to outbound trains such that every outbound train can be built on a separate track, and where individual cars of different trains can temporarily be stored on a special subset of the tracks, so called mixed tracks. We

observe that the core problem can be formulated as a special list coloring problem in interval graphs, which is known to be NP-complete. The usage of mixed tracks induces several new variants of the list-coloring problem in which the given intervals can be shortened by cutting off a prefix of the interval. We show that in case of uniform and sufficient track lengths, the corresponding coloring problem can be solved in polynomial time, if the goal is to minimize the total cost associated with cutting off prefixes of the intervals. Based on these results, we devise two heuristics as well as an integer program to tackle the problem. As a case study, we consider a real-world problem instance from the Hallsberg Rangerbangård classification yard in Sweden. Planning over horizons of seven days, we obtain feasible solutions from the integer program in all scenarios, and from the heuristics in most scenarios.

Sorting Trains at Classification Yards. Another problem at classification yards is that of sorting cars of outbound trains. Cars are sorted such that when a group of cars reaches its final destination, this group is at the end of the train and can thus be easily decoupled without further shunting operations. Here, we answer an open question regarding the complexity of sorting trains in a hump yard when the order of arrival of inbound trains can be chosen.

Theoretical Models for Dispatching. The third part of this thesis deals with theoretical problems that are related to planning problems occurring during daily railway operations. In the presence of delays, a dispatcher has to make decisions to return to the planned timetable as close as possible. This is a very challenging area, for which there are no fully automatized solutions yet that work on large scale networks. In this part, we seek to better understand the combinatorial nature underlying such problems, to analyze their complexity, and to design polynomial time or approximation algorithms. For this theoretical work, we do not carry out experiments, as the abstractions we make neglect too many details of railway operations.

Selecting Vertex Disjoint Paths in Planar Graphs. This work is motivated by operational planning problems, such as rerouting and rescheduling trains in the presence of delays. As an abstraction, we study variants of the vertex disjoint paths problem in planar graphs where paths have to be selected from given sets of paths. We investigate the problem as a decision, maximization, and routing-in-

rounds problem. Although all considered variants are NP-hard in planar graphs, certain restrictions on the location of the terminals on the outer face of the planar embedding of the graph lead to polynomially solvable cases for the decision and maximization versions of the problem and to an approximation algorithm for the routing-in-rounds problem.

MIS in Outersegment Graphs. The latter work led us to the open question of the complexity of finding a maximum independent set in outerstring graphs. While this interesting question from graph theory is still open, we were able to show that the problem is solvable in polynomial time for a subclass of outerstring graphs, namely outersegment graphs. An outersegment graph is the intersection graph of line-segments lying inside a disk and having one end-point on the boundary of the disk. We present a polynomial-time algorithm for the problem of computing a maximum independent set in outersegment graphs where every segment is either horizontally or vertically aligned. We assume that a geometric representation of the graph is given as input.

Assigning Move-up Crews. A delayed train necessarily delays its crew. To prevent the propagation of delay to the crew's next trip, a so called move-up crew may take over the rest of the delayed crew's duty. We address a theoretical abstraction of the problem of making optimal crew swap decisions during operations. We give efficient algorithms for the local case and show that optimizing crew swaps over the whole railway network is NP-hard.

Part I

Utilizing Delay Data

Chapter 2

Scheduling Additional Trains in Dense Corridors

2.1 Introduction

Since the introduction of the railway development plan Rail 2000 in Switzerland [84], the demand for passenger train transportation has steadily been increasing. As a consequence, Swiss Federal Railways (SBB) offers more trains. It seems difficult or even impossible to expand track resources at the same rate as passenger numbers and the demand for higher train frequencies increases. Therefore, railway traffic is becoming denser, making both resource scheduling and delay management more difficult and of major importance.

In this chapter, we address the recurring problem of adding a train path, i.e., a schedule for a single train in terms of track allocation in space and time, on a given dense corridor, i.e., an important subnetwork in form of a path between two major stations. In particular, we are interested in finding *robust* train paths, i.e., those which entail a low risk for the additional train of being delayed upon arrival at the final station. Currently, planners use a mixture of domain knowledge and past experience to come up with potential solutions which then undergo detailed simulations to select the most appropriate solution.

We present a model that supports railway planners by computing a set of recommended train paths for a given train request.

A novelty of this chapter is our approach to obtain such recommendations: we use extensive historic delay data of SBB to compute these recommendations. The underlying data has been recorded by SBB during the operation of recent timetables. We combine risk predictions with a combinatorial model that can answer the planners' queries very quickly. As there is a trade-off between risk of delay and travel time of a train path, not only a single solution is computed, but we compute the Pareto frontier of solutions with respect to travel time and expected delay of the additional train upon arrival at its final station. Thus, as an advantage over simulating just a few scenarios, the planners get a range of different, Pareto optimal solutions. Another advantage is that most of the necessary data are available from the database. In contrast, collecting the necessary data for detailed simulations can be a labour intensive task.

The data provided by SBB are recorded by track vacancy systems, which are part of the operational security system. This system registers the number and actual passing time of every train at several thousand points in the network, resulting in an enormous amount of data. These data are aggregated to about 2300 operating points or stations that SBB consider most important, and are stored permanently in a database. SBB planners use these data to monitor the quality of the current timetable, to detect recurrent deviations of trains from their planned timetable and to improve future timetables. The data implicitly contain a wealth of information, e.g., dependencies between trains, resource bottlenecks, or dispatching decisions. We will show how to profit from these information.

This chapter is structured as follows. In Section 2.1.1 we present the problem more formally. Our solution approach consists of two main steps. In the first step, we extract predictors from the historic delay data in order to compute a series of linear regression models for risk prediction. This is detailed in Section 2.2. In the second step, an algorithm is executed to find a Pareto optimal set of train paths for a given request, using the risk prediction of the first step. We give the algorithmic details and complexity results of the second step in Section 2.3. In Section 2.4 we present experiments that show the quality of our approach. We discuss a certain bias resulting from our algorithm in Section 2.5. Finally, conclusions and open problems are

given in Section 2.6.

2.1.1 Problem Description

A typical client request which planners have to deal with is, e.g., "add one train in the morning rush hour between Bern and Zurich". More formally, a train request r specifies a corridor, i.e., a sequence of stations and operating points $\langle S_1, S_2, \ldots, S_\ell \rangle$ in the network, the type $\vartheta(\tau)$ of the additional train τ (e.g. local, regional, long distance), the dates on which the train should run (e.g. weekdays, weekends), earliest and latest time of departure $[\underline{d}, \overline{d}]$ at S_1, earliest and latest time of arrival $[\underline{a}, \overline{a}]$ at S_ℓ, as well as intermediate stops, if any, at stations along the corridor. Given the request, the planner has to add an additional train from station S_1 to station S_ℓ on the corridor to a timetable that has been in operation over a period of time.

The planner's task is to find a *train path* π that satisfies the client's request and has a low risk of delay upon arrival of the train at the final station. A train path is characterized by the arrival times a_i and departure times d_i at station S_i, $i \in \{1, \ldots, \ell\}$ and by pass-through times p_i at stations (or operating points) where the train does not stop, e.g., $\pi = (d_1, p_2, p_3, a_4, d_4, p_5, a_6)$. For brevity, we will refer to operating points as stations, even though most operating points are placed on the tracks.

2.1.2 Summary of Results

We propose a new method to predict the risk of delay of a planned train using linear regression models on the basis of extensive real-world delay data of trains. We suggest several predictors that can be computed from historic delay data and how to compute linear regression models for the stations along a given corridor. We show how to integrate these models into a combinatorial shortest path model. We give a practically efficient algorithm to compute a set of Pareto optimal train schedules with respect to risk and travel time. We discuss the choice of different models and notions of risk with respect to the algorithmic complexity of the resulting combinatorial problems. Finally, we demonstrate the quality of our models on real-world data of Swiss Federal Railways.

2.2 Regression Model

In this section, we describe how to compute linear regression models to predict the delay of a planned train path between two consecutive stations of a corridor. We will use these models in the next section in order to predict the delay of of a planned train path along a whole corridor. We will use this prediction for our notion of *risk*, which will also be covered in the next section.

For the statistical terminology used in this section, see any textbook on linear models, for example [38]. In short, a linear regression model has the form

$$Y = \alpha + X\beta + \varepsilon,$$

where Y is a vector of dependent variables (the delay that we seek to predict), X is a matrix of predictor variables (values that are computed from the delay data, one row per day), β are the regression coefficients (the result of computing the linear model), and ε are the error terms.

Our general approach is to first extract potential predictors for the delay of a specific train path at a specific time and station from the recorded delay data. Using these predictors, a series of linear regression models is computed. These models allow us to predict delays of *historic* train paths during the period in which the current timetable has been operational. Thus, the predicted delays are not in the future but in the past! The purpose of our prediction is not to be able to predict delays of current trains in the future, but to be able to evaluate how an additional train would have been delayed if it had run on a specific train path on a past day of the current timetable period. We finally define the risk of a train path as an aggregated value of the delay predictions for all days of the recorded period. This definition allows to associate a risk with every feasible train path. An appropriate choice of the prediction model will allow us to search efficiently for a set of Pareto optimal paths by a special shortest path computation in a time expanded graph.

2.2.1 Predictors and Linear Regression Models

The first step towards prediction of delays and thus the final goal of a conclusive risk measure is to identify relevant predictors that can be extracted from the recorded delay data. In cooperation with planners

from SBB we identified the following potential "causes" for the delay of a train τ on train path π upon arrival at station S_i on day d, with planned arrival time a_i:

previous delay
: the delay $\delta_{i-1}(\tau, d, \pi)$ of train τ upon departure from the previous station S_{i-1} on day d, which may propagate to S_i

properties of the train
: a set of indicator variables $\vartheta(\tau)$, one for each possible train type (e.g., local train, high-speed train, etc.)

actual traffic density
: the number of actual train arrivals and departures at S_i, denoted by $w_q(a_i, d)$, $q \in (\mathcal{I} \times \mathcal{J})$, for a set of time intervals $\mathcal{I}$ (windows) around the planned arrival time a_i, and a set $\mathcal{J}$ of four cases distinguishing between trains arriving or departing in the same or opposite direction as train τ

planned traffic density
: the time difference $\Delta_j^{\text{prev}}(a_i, d)$ to the planned arrival of the train that is scheduled to be the j-th train arriving before train τ at S_i (and driving in the same direction as τ); similarly, $\Delta_j^{\text{next}}(a_i, d)$ is defined as the j-th train planned to arrive after train τ

planned slack times
: a slack time of $s(\tau, d_{i-1}, a_i)$, i.e., the difference between the planned travel time and the minimum driving time of train τ between stations S_{i-1} and S_i;

delays of neighbor trains
: delays of the trains that are scheduled around the planned arrival time a_i, e.g., $\delta_4^{\text{prev}}(a_i, d)$ denotes the delay of the train that was planned to be the fourth train arriving before a_i on day d among the trains driving in the same direction as τ

track properties
: the average net change in delay between S_i and S_{i-1} during one hour around a_i (± 30 minutes) on day d, denoted by $l(a_i, d)$

Figure 2.1: Sequence of models where the delay predicted by the previous model is a predictor for the next.

Our goal is to use the most relevant of the above predictors in the linear regression models. We emphasize that we are not mainly interested in the exact type of dependence of predictors and dependent variable but rather in a model that predicts well and that blends well with our combinatorial search for a low risk path.

To get meaningful models with well-balanced bias and variance we select a subset of predictors that lead to models with high Akaike information criterion (AIC) [5]. The AIC is an established tool for model selection that tends to avoid overfitting problems. To find these models we used the greedy stepAIC algorithm of Venables and Ripley [82] implemented in R [83]. We stopped this algorithm after 20 steps when usually no significant further improvement in terms of AIC was made. We note that due to the large amount of available data, overfitting is not very likely to occur in our case even if we include the full set of predictors. Given the stations $\{S_1, \ldots, S_\ell\}$ along the corridor, we set up a linear regression model $\mathcal{M}_{i-1,i}$ for each pair of consecutive stations (S_{i-1}, S_i), $i \in \{2, \ldots, \ell\}$. We call these models the *between-stations models*. We also set up an *in-station model* $\mathcal{M}_{i,i}$ for each intermediate station S_i in which the train is requested to stop, by analogous definitions of predictors. Model $\mathcal{M}_{i-1,i}$ uses the set of predictors sketched above, in particular the previous delay as predicted by model $\mathcal{M}_{i-2,i-1}$ (or $\mathcal{M}_{i-1,i-1}$). This means that the prediction of the last model is used as predictor for the next model in the sequence. The dependency of the models is illustrated in Figure 2.1.

The models $\mathcal{M}_{i-1,i}$ are basically of the form

$$\begin{aligned}
\delta_i(\tau,d,\pi) &= \alpha+\beta_1\delta_{i-1}(\tau,d,\pi)+\beta_2\vartheta(\tau)\\
&\quad+\sum_q \beta_{3,q}w_q(a_i,d)\\
&\quad+\sum_j \left(\beta_{4,j}\Delta_j^{\text{prev}}(a_i,d)+\beta_{5,j}\delta_j^{\text{prev}}(a_i,d)\right)\\
&\quad+\sum_{j'} \left(\beta_{6,j'}\Delta_{j'}^{\text{next}}(a_i,d)+\beta_{7,j'}\delta_{j'}^{\text{next}}(a_i,d)\right)\\
&\quad+\beta_\ell l(a_i,d)+\beta_m s(\tau,d_{i-1},a_i)\\
&\quad+\varepsilon_{i,\tau,d}\\
&= \text{model}(\tau,a_i,d_{i-1},d,\delta_{i-1})+\varepsilon_{i,\tau,d} \qquad (2.1)
\end{aligned}$$

using dummy-variable regression for the categorical predictor $\vartheta(\tau)$. We make the usual assumptions for linear regression models concerning the errors $\varepsilon_{i,\tau,d}$, namely linearity, constant variance (homoscedasticity), normality, and independence:

$$\begin{aligned}
\mathbb{E}(\varepsilon_{i,\tau,d}) &= 0,\\
\text{Var}(\varepsilon_{i,\tau,d}) &= \sigma^2,\\
\varepsilon_{i,\tau,d} &\sim N(0,\sigma^2), \text{and}\\
\text{Cov}(\varepsilon_{i,\tau,d},\varepsilon_{i',\tau',d'}) &= 0.
\end{aligned}$$

Note the dependency of the fitted value for $\hat{\delta}_i(\tau,d,\pi)$ on τ, a_i, d_{i-1}, d, and $\delta_{i-1}(\tau,d,\pi)$ as indicated by the term model$(\ldots)$.

We will show in Sections 2.3 and 2.4.1 that our choice of a sequence of *linear regression models* leads indeed to surprisingly accurate predictions of past delays.

2.2.2 Series of Regression Models

In order to compute a risk measure for a train path $\pi=(d_1,\ldots,a_\ell)$ we use the sequence of regression models $(\mathcal{M}_{1,2},\ldots,\mathcal{M}_{\ell-1,\ell})$ to predict the delay of each day d for the recorded period of time.

$$\hat{\delta}_i(\tau,d,\pi)=\begin{cases}\text{model}(\tau,a_i,d_{i-1},d,\hat{\delta}_{i-1}) & \forall i>1\\ \delta_0(\tau,d,\pi)\end{cases} \qquad (2.2)$$

where $\delta_0(\tau, d, \pi)$ is an estimation of the start delay of the train (for example the average delay of trains in that hour of the day).

As a risk measure we propose an aggregated value of these values:

Definition 2.1 (risk)**.** *For a given train τ and a train path π we define its* risk *with respect to a recorded period D and a prediction model as risk$(\pi) = \frac{1}{|D|}\sum_{d\in D}\hat{\delta}_\ell(\tau, d, \pi)$, where the $\hat{\delta}_\ell(\tau, d, \pi)$ values are obtained via the regression models as in (2.2).*

There are different types of possible regression models. In particular, if we restrict the model above to a subset of the predictors, we can limit its dependency on the data. A very basic model depends only on $\hat{\delta}_{i-1}, a_i, \tau$ and d. Such a "*basic*" model could look as follows:

$$\begin{aligned}\hat{\delta}_i(\tau, d, \pi) &= \text{model}(a_i, \tau, d, \hat{\delta}_{i-1})\\ &= \alpha + \beta_1\hat{\delta}_{i-1} + \beta_2\vartheta(\tau) + \sum_q \beta_{3,q} w_q(a_i, d)\\ &\quad + \sum_j \left(\beta_{4,j}\Delta_j^{\text{prev}}(a_i, d) + \beta_{5,j}\delta_j^{\text{prev}}(a_i, d)\right)\\ &\quad + \sum_{j'} \left(\beta_{6,j'}\Delta_{j'}^{\text{next}}(a_i, d) + \beta_{7,j'}\delta_{j'}^{\text{next}}(a_i, d)\right)\\ &= \beta_1\hat{\delta}_{i-1} + q(a_i, \tau, d) \qquad (2.3)\end{aligned}$$

Here $q()$ is a value that depends only on the indicated terms. More advanced models depend also on d_{i-1}, use power transformed predictors, or involve interaction terms not containing $\hat{\delta}_{i-1}$. Interaction terms are basically products of predictors, see [38] for more details. Such models can for example take into consideration the interaction between track loss and slack, as in the following "*advanced*" model:

$$\begin{aligned}\hat{\delta}_i(\Theta, d, \pi) &= \text{model}(\tau, a_i, d_{i-1}, d, \hat{\delta}_{i-1})\\ &= \text{basic (2.3)}\\ &\quad + \beta_8 l(a_i, d) + \beta_9 s(\tau, d_{i-1}, a_i)\\ &\quad + \beta_{10} l(a_i, d) : s(\tau, d_{i-1}, a_i) + \ldots\\ &= \beta_1\hat{\delta}_{i-1} + q(a_i, d_{i-1}, \tau, d) \qquad (2.4)\end{aligned}$$

This could model the potential situation that trains with high slack between two stations are not affected by high track losses of other trains, whereas trains with low slack are.

If one wants to model that different types of train can catch up differently on delays one would also have to include interaction terms involving $\hat{\delta}_{i-1}$. An example would be the idea that track loss and previous delay interact, i.e., in situations with high track loss a high previous delay will lead to a high delay at the current station, whereas with low track loss it will have a much smaller effect. Such a model of type "*all interactions*" may look like:

$$\begin{aligned}\hat{\delta}_i(\Theta, d, \pi) &= \text{advanced (2.4)} \ + \ldots + \hat{\delta}_{i-1} : \vartheta(\tau) \\ &= p(a_i, d_{i-1}, \tau, d)\hat{\delta}_{i-1} + q(a_i, d_{i-1}, \tau, d) \end{aligned} \tag{2.5}$$

One critique of these models could be that the time windows are fixed around the *planned* arrival time of a train. In some delay scenarios, however, it is clear that the train will arrive with large delay, so that one would prefer to have the time windows around a later "approximate" arrival time rather than around the earlier planned time. For that reason, one could make predictors dependent on an estimated arrival time as in the following example:

$$\begin{aligned}\hat{\delta}_i(\Theta, d, \pi) &= \text{all interactions (2.5)} \ + \ldots \\ &\quad + w'(a_i + 0.8 \cdot (\hat{\delta}_{i-1} - d_{i-1}), d) \\ &= p(a_i, d_{i-1}, \tau, d, \hat{\delta}_{i-1})\hat{\delta}_{i-1} + q(a_i, d_{i-1}, \tau, d, \hat{\delta}_{i-1}) \\ &= f(a_i, d_{i-1}, \tau, d, \hat{\delta}_{i-1}) \end{aligned} \tag{2.6}$$

This model differs from the previous models in that for constant (a_i, d_{i-1}, τ, d), i.e., for a given train and train path, the previous models all boil down to a simple linear function in $\hat{\delta}_{i-1}$. The last model, however, can be an "arbitrary" function in $\hat{\delta}_{i-1}$ in this situation.

2.3 Shortest Path Algorithms

The search for train paths with low risk as outlined in the last section leads to a shortest path problem on an appropriately defined time expanded graph. In the following, we describe how this graph is constructed, discuss the algorithmic complexity of possible shortest path models, and describe the algorithm used in our experiments.

2.3.1 Time Expanded Graph Model

We want to construct a graph such that every path in the graph corresponds to a feasible planned train path. We restrict ourselves to the following important constraints: realistic driving times, the number of available parallel (bidirectional) tracks between stations, and headway times, i.e., the security requirement that a train can follow another one on the same track only after a certain time span. Given a train request r, a layered, time expanded graph $G_r = (V_1 \uplus V_2 \uplus \ldots \uplus V_\ell, E)$ is constructed as follows. Each node $v_i^t \in V_i$ represents a station S_i at a certain point in time t. The earliest departure time $\underline{d}$ at the first station S_1 and latest arrival time $\overline{a}$ at the last station S_ℓ of the corridor define, together with the train type $\vartheta(\tau)$, feasible time windows for each layer. Within each such time window, nodes are created according to a certain granularity, e.g., 10 nodes per minute. Every edge $\left(v_i^t, v_{i+1}^{t'}\right) \in E$ represents a driving activity between two stations. A dwelling activity within a station is represented by an edge $\left(v_i^t, v_i^{t'}\right)$. For simplicity, we denote v_i^t simply by v_i and the edge $\left(v_i^t, v_{i+1}^{t'}\right)$ by $e_{i,i+1}$ when the points in time to which a specific node refers are unambiguous. Thus, every v_1-v_ℓ path in G_r with $v_1 \in V_1, v_\ell \in V_\ell$, corresponds to a train path π.

To model realistic driving times, we distinguish between three types of nodes representing the state of the train, namely *arrival* (arr), *departure* (dep), and *pass-through* (pass) nodes. G_r may only contain edges between certain types of nodes as shown in Figure 2.2, and which further respect minimum and maximum driving times according to the train type $\vartheta(\tau)$.

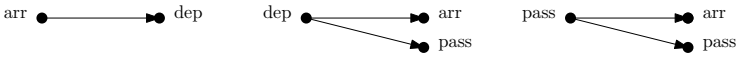


Figure 2.2: G_r may only contain edges between the following types of nodes: arr $\rightarrow$ dep, both nodes belonging to the same station, and $\{\text{dep} \mid \text{pass}\} \rightarrow \{\text{arr} \mid \text{pass}\}$, where head and tail nodes belong to consecutive stations.

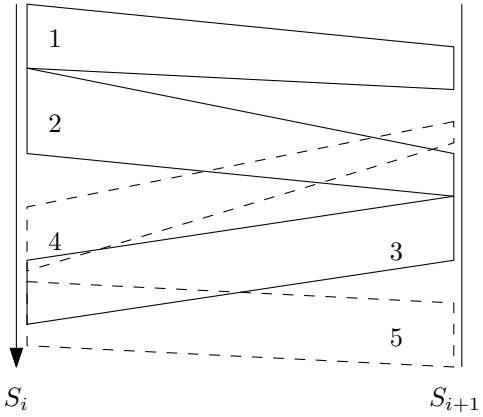
Track capacities and headway constraints are modeled by omitting those edges of G_r which would cause a train path to be infeasible w.r.t. the current schedule. Hence, for every potential edge we

need to decide if, given a certain number of parallel tracks, it would be possible to schedule an additional train on the track segment at the time specified by that edge.

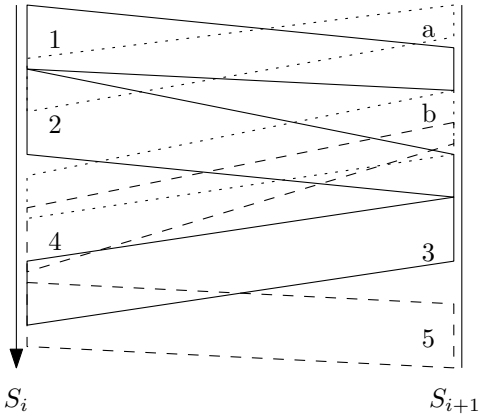
For our puposes, the time during which a track is blocked by a train can be modeled by a trapezoid, as shown in Figure 2.3(a). To compute whether an additional train could be added to the timetable, given the limited number of tracks, we make use of trapezoid graphs, i.e., intersection graphs of trapezoids, see [18, 25]. The problem of deciding whether an edge $e_{i,i+1}$ is feasible with regard to track capacity reduces to the chromatic number problem in trapezoid graphs as follows. Take the trapezoidal representation of all scheduled trains and add the trapezoid corresponding to $e_{i,i+1}$, i.e., of the additional train. If and only if the chromatic number of the corresponding trapezoid graph is not greater than the number of available parallel tracks, then the edge is feasible and can be added to G_r. Figures 2.3(b) and (c) give an example. The chromatic number problem for trapezoid graphs can be solved in time $O(n \log n)$, see [25]. The question is now how many of the already scheduled trains have to be considered to decide upon the feasibility of an edge of G_r. Because trapezoid graphs are perfect graphs, the chromatic number equals the size of a maximum clique. But as we know that the already scheduled trains constitute a feasible solution, i.e., do not use more tracks than are available, we only have to check whether the new instance has a larger maximum clique. Clearly, the latter can be reduced to computing the size of the largest clique containing the new trapezoid.

2.3.2 Model Choice and Algorithmic Complexity

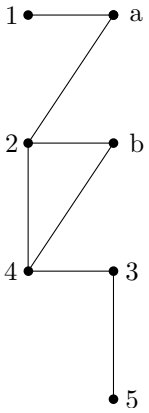
In this section we discuss the complexity of several shortest path problems arising from different regression models. As planners are not only interested in minimizing the risk of delay, but also in keeping the planned travel time reasonably short, we need to compute a set of Pareto optimal paths (w.r.t. risk and travel time) in G_r. Apart from the planned travel time, we need to assign *costs* to paths, reflecting their risk according to Definition 2.1. First, we consider the cost of a path for a single day $d \in D$ only. In this case, we let the risk equal the predicted delay $\hat{\delta}_\ell(\tau, d, \pi)$. The structure of Equations 2.3, 2.4, and 2.5 leads to a cost structure, in which on each edge $e_{i,i+1}$ the accumulated delay $\hat{\delta}_i(\tau, d, \pi)$ at v_i is multiplied with a constant



(a)



(b)



(c)

Figure 2.3: (a) Trapezoidal representation of a schedule for five trains between two consecutive stations S_i and S_{i+1}. Each trapezoid represents the resource utilization (headway constraints) of a single train in time (vertical axis) and space (horizontal axis). Hence train 2 drives from S_i to S_{i+1}. Here, two tracks suffice, as trains 1, 2, and 3 can be scheduled on one track and trains 4 and 5 on the other without conflict. (b) Requests for additional trains a and b. (c) Corresponding trapezoid graph. Trains 1 to 5 and the additional train a could still be scheduled on two parallel tracks, whereas to add train b, three parallel tracks would be necessary (since the respective maximum cliques of a and b have size two and three, respectively).

$p(e_{i,i+1})$ and then a constant $q(e_{i,i+1})$ is added to this value to yield $\hat{\delta}_{i+1}(\tau, d, \pi)$. Hence, to define the cost structure for the minimum risk computation for a single day, one can annotate the edges with these pairs (p, q). More formally, mirroring Equation 2.2, the cost of a path $\pi = (v_1, v_2, \ldots, v_\ell)$ in G_r can be recursively defined as follows:

$$\text{cost}(v_1, v_2, \ldots, v_i) = \begin{cases} p(v_{i-1}, v_i)\text{cost}(v_1, v_2, \ldots, v_{i-1}) + q(v_{i-1}, v_i) & \text{for } i > 1 \\ \delta_0(\tau, d, \pi) & \text{for } i = 1 \end{cases} \tag{2.7}$$

which yields

$$\begin{aligned} \text{cost}(\pi) = \delta_0(\tau, d, \pi)p(e_{1,2})p(e_{2,3}) \cdots p(e_{\ell-1,\ell}) \\ + q(e_{1,2})p(e_{2,3}) \cdots p(e_{\ell-1,\ell}) \\ + \ldots + q(e_{\ell-2,\ell-1})p(e_{\ell-1,\ell}) + q(e_{\ell-1,\ell}) \end{aligned} \tag{2.8}$$

The risk computation for the whole period D can be carried out by doing the above computation for each day $d \in D$ resulting in $|D|$ delay predictions, which can be read as a vector $(\hat{\delta}_\ell(\tau, 1, \pi), \ldots, \hat{\delta}_\ell(\tau, |D|, \pi))^T$. According to Definition 2.1, the risk is the average over the entries of this vector. It follows that the full shortest path problem is a problem over vectors of dimension $|D|$, which we formalize in the following definition.

Definition 2.2. *Given a layered, time-expanded graph $G_r = (V_1 \uplus V_2 \uplus \ldots \uplus V_\ell, E)$ with edges $e = (v_i, v_{i+1})$ labeled by $(p(e), q(e))$. The* one day minimum risk problem *asks for a path from layer 1 to layer ℓ of minimum cost at layer ℓ, where the cost of a path π is computed according to Equation 2.8 recursively along the path. In the more general* minimum risk problem *edges are annotated with pairs of $|D|$-dimensional vectors $(\vec{p}(e), \vec{q}(e))$ instead of scalars. For a given path π its cost is computed as the average over the costs $c_i, 1 \leq i \leq |D|$ for the components, where the cost of a component is again computed according to (2.8) for each component separately.*

With these definitions, we are ready to discuss the complexity of various problem variants:

Theorem 2.3. *As long as the prediction functions model() are monotone increasing in $\hat{\delta}_{i-1}$, the one day minimum risk problem can be solved in polynomial time by a label setting algorithm.*

Proof. For a proof, it suffices to note that the subpath optimality property of shortest paths [3, Property 4.1] holds for such prediction functions. Therefore, a label setting algorithm that updates labels in a topological order is correct; the correctness proof in [3, Section 4.4] can be applied one to one. □

This theorem characterizes in a sense "well-behaved" models. If the predictions of a model are not monotonically increasing in $\hat{\delta}_{i-1}$ then a model might predict that a train arrives earlier at station i for larger delays at station $i-1$. Note that models of type (2.5) (where predictors may depend on an estimated arrival time) can have exactly this behavior.

For an efficient algorithm for the minimum risk problem we need more than just the efficient computation of the one day problem.

Theorem 2.4. *If all components of the cost vectors $\vec{p}(e_{i,i+1})$ are equal to a single value $p_{i,i+1}$ for each layer $1 \leq i < \ell - 1$ of G_r, i.e., $\vec{p}(e_{i,i+1}) = p_{i,i+1}\mathbf{1}$, then the minimum risk problem can be solved by a label setting algorithm in time polynomial in the size of the time expanded graph G_r.*

Proof. Denote the cost of path $\pi = (v_1, v_2, \ldots, v_\ell)$ on day $d \in D$ by $\text{cost}_d(\pi)$. For an edge $e = (v_{i-1}, v_i)$ and day d we write for the respective components of $\vec{p}(e)$ and $\vec{q}(e)$ the terms $p_d(e) = p_{i-1,i}$ and $q_d(e)$. The proof is by induction on subpaths of π of length i. Consider now a subpath $(v_1, v_2, \ldots, v_i)$ of length $i > 1$ of π.

$$
\begin{aligned}
&\text{risk}(v_1, v_2, \ldots, v_i) \\
&= \frac{1}{|D|} \sum_{d \in D} \text{cost}_d(v_1, v_2, \ldots, v_i) \\
&= \frac{1}{|D|} \sum_{d \in D} p_{i-1,i} \cdot \text{cost}_d(v_1, v_2, \ldots, v_{i-1}) + q_d(v_{i-1}, v_i) \\
&= p_{i-1,i} \cdot \text{risk}(v_1, v_2, \ldots, v_{i-1}) + \bar{q}(v_{i-1}, v_i) \qquad (2.9)
\end{aligned}
$$

Here $\bar{q}(e)$ denotes the average $\frac{1}{|D|} \sum_{d \in D} q_d(e)$ of the components of $\vec{q}(e)$. The last equation is exactly of the form (2.7) for a single day

instance with edge costs $p_{i-1,i}$, $\bar{q}(e_{i-1,i})$ for each edge $e_{i-1,i} \in E$. The same holds for $i = 1$. Therefore, by Theorem 2.3 the problem can be solved by a label setting algorithm in polynomial time. □

Fortunately, this condition is met by the "basic" models (2.3), by the "advanced" models (2.4) and even by models that include interactions of $\hat{\delta}_{i-1}$ and predictors like $\vartheta(\tau)$ or $s(\tau, d_{i-1}, a_i)$ that do not depend on d. The above theorem is complemented with an NP-hardness proof for models with varying $\vec{p}$.

Theorem 2.5. *The general minimum risk problem (without the condition of Theorem 2.4 on the $\vec{p}$ vectors) is NP-hard.*

Proof. We show a reduction from the NP-complete set cover problem [41, SP5]: Given a ground set U of n elements and a collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of subsets of U, is there a subcollection $\mathcal{C}'$ of $\mathcal{C}$ with cardinality k that covers all elements of U? Given such an instance, we create a minimum risk problem instance as depicted in Figure 2.4: As for the graph G_r, it consists of $m + 1$ layers $i \in \{1, \ldots, m + 1\}$, each consisting of a single node v_i. Each pair of consecutive layers $i, i + 1$ is connected by two parallel (top and bottom) edges $\overline{e}_{i,i+1}, \underline{e}_{i,i+1}$ (note that one could easily replace parallel edges by splitting each edge into two by a single node). We let the cost vectors $\vec{p}$ and $\vec{q}$ have $n + 1$ components. The first n components of $\vec{p}(\overline{e}_{i,i+1})$ are the indicator vectors of set C_i, the last one is always 1. All $\vec{q}(\overline{e}_{i,i+1})$ are set to $(0, \ldots, 0, 1)^T$, all $\vec{p}(\underline{e}_{i,i+1})$ are set to the all one vector, all $\vec{q}(\underline{e}_{i,i+1})$ are set to the all zero vector. Finally, the initial delays δ_0 are set to $k + 1$ for the first n components and to 0 for the last. We claim that there is a solution to the set cover problem if and only if there is a solution to the min risk problem of cost at most $\frac{k}{n+1}$. From the construction there is a bijection of subcollections $\mathcal{C} = \{C_{i_1}, \ldots, C_{i_{k'}}\}$ and paths through the graph that take the top edges exactly at layers $(i_1, i_1+1), \ldots, (i_{k'}, i_{k'}+1)$ and have cost $\left(k' + (k+1)\left|U \setminus \bigcup_{1 \le j \le k'} C_{i_j}\right|\right)/(n+1)$. This bijection directly gives the theorem. □

This concerns models of type "all interactions" (2.5) that include for example interaction terms of $\hat{\delta}_{i-1}$ and some window variables or any other predictor that depends on d.

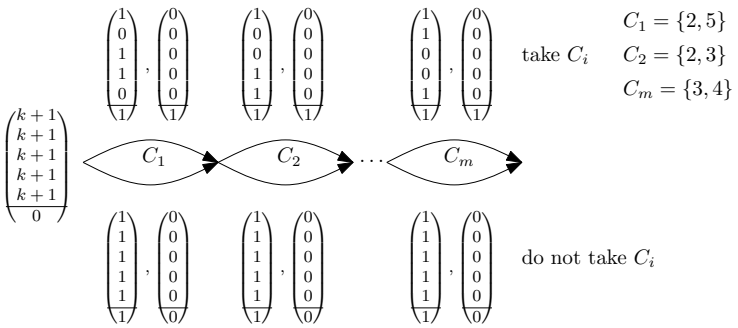


Figure 2.4: Transformation from set cover. The costs on the arcs are indicated as $(\vec{p}, \vec{q})$

As far as the aggregation function in the risk computation is concerned, SBB planners prefer to work with the more robust median. Therefore, one could also define

$$\widetilde{\text{risk}}(\pi) = \text{median}\left\{\hat{\delta}_\ell(\tau, d, \pi) \mid d \in D\right\}, \tag{2.10}$$

i.e., the median of the delay predictions for the last station. This choice, however, leads to an NP-hard shortest path problem already for the case where the cost of a path is simply the sum of its edge weights (which are vectors), as the following theorem shows.

Theorem 2.6. *For the median as an aggregation function in the risk computation the classical shortest path problem with respect to this cost measure $\widetilde{risk}(\pi)$ is NP-hard already for additive vector valued edge costs and therefore also for all variants discussed here.*

Proof. We show a reduction from the weakly NP-complete partition problem [41][SP12]: Given a finite set $U = \{u_1, \ldots, u_n\}$ and a size $s(u_i)$ for each element $u_i \in U$, and let $s(U') := \sum_{u_i \in U'} s(u_i)$ for all $U' \subseteq U$, is there a partition of U into sets U_1 and U_2 such that $s(U_1) = s(U_2)$? The proof is along the lines of Figure 2.5: We use three dimensional vectors, the last component of which is always a large number $M > s(U) := \sum_{1 \le i \le n} s(u_i)$ denoted by ∞ in the figure. The construction is similar to the one in the proof of Theorem 2.5. We set up a graph of $n + 1$ layers, again with parallel top and bottom edges. This time the edges $\overline{e}_{i,i+1}$ and $\underline{e}_{i,i+1}$ have single

cost vectors as indicated in the figure. There is a bijection of partitions of U into U_1 and U_2 with $x = \max\{s(U_1), s(U_2)\}$ and paths in G that take the top edge exactly at the elements of U_1 and have a median cost of x. Again, this bijection directly gives the theorem. □

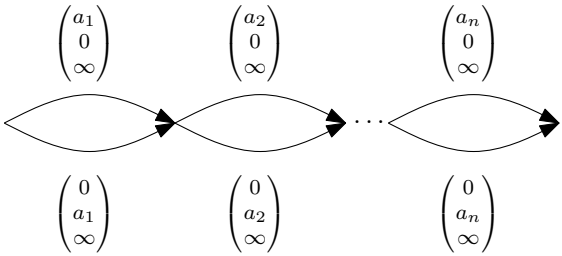


Figure 2.5: Transformation from partition.

As every v_1-v_ℓ path is only a suggestion for the planner, who may have to take further feasibility requirements into account, we would like to provide a set of k "best" solutions. Calculating the k shortest paths, however, would lead to a set of solutions that are very similar to each other. Instead, we propose to compute the Pareto frontier with respect to the trade-off between risk and travel time, which is a natural choice in this context. Hence, for each feasible travel time from S_1 to S_ℓ, if a path with such a travel time exists, then the Pareto frontier will contain such a path of minimum risk.

Lemma 2.7. *The size of the Pareto frontier is proportional to the size of the requested departure and arrival time windows.*

Proof. Given the departure time window $[\underline{d}, \overline{d}]$ and arrival time window $[\underline{a}, \overline{a}]$, then the travel time of any v_1-v_ℓ path in G_r can be bounded from above by $\overline{a} - \underline{d}$, and from below by $\underline{a} - \overline{d}$. As time is discretized to a fixed granularity of, say, g seconds, there are at most $\lceil \frac{\overline{a}-\underline{a}+\overline{d}-\underline{d}}{g} \rceil$ different possible travel times and hence elements in the Pareto frontier. □

Hence, the size of the Pareto frontier is only pseudo-polynomial in the size of the input. This is, however, not an obstacle, since the range of possible travel times is limited in practice.

2.3.3 Algorithm

We sketch the algorithm to find a set of Pareto optimal v_1-v_ℓ paths in G_r. As G_r is acyclic, it suffices to consider each edge once in the order given by any topological sorting of the nodes and to apply a reaching algorithm [3]. Note that the topological sorting of the nodes is readily available by the order of the stations along the corridor, as each node is associated with one station.

Algorithm 1: Compute Pareto optimal Paths

Input: User request r, models $\mathcal{M}_{i,i+1}$ and $\mathcal{M}_{i,i}$, $\forall\, i \in \{1, \ldots, \ell - 1\}$
Output: Pareto frontier F of v_1-v_ℓ paths

```
Create Graph G_r;
Initialize F to contain a v_1-v_1 path for each v_1 ∈ V_1^dep;
for i ← 1 to ℓ do
    foreach v_i ∈ V_i^arr do                    /* in station */
        updateNeighbors(v_i, M_{i,i})
    end
    foreach v_i ∈ V_i^dep ∪ V_i^pass do          /* betw. stations */
        updateNeighbors(v_i, M_{i,i+1})
    end
end
foreach v_ℓ ∈ V_ℓ do
    F ←insert(F, F(v_ℓ))
end
```

In Algorithm 1, G_r is created as defined above. Recall that we write v_i for a node v_i^t (of station S_i at time t). For performance reasons, all nodes and edges that are not on a v_1-v_ℓ path can be removed in a preprocessing step. Associated with every node v_i is a Pareto frontier $F(v_i)$ of paths from S_1 to S_i. The procedure updateNeighbors(v_i, $\mathcal{M}_{i,j}$) checks for each edge (v_i, v_j), in the adjacency list of v_i, whether the paths in $F(v_i)$ can be extended to v_j, such that they dominate paths in $F(v_j)$. If this is the case, the new path is inserted into $F(v_j)$. The procedure insert($F, F(v_\ell)$) inserts each path in $F(v_\ell)$ into the final Pareto Frontier F, if possible. This is done for all Pareto frontiers of the nodes v_ℓ at the last station S_ℓ.

Thus, F contains only Pareto optimal v_1-v_ℓ paths.

Even though the estimators obtained from the linear regression models are unbiased, this unbiasedness is lost in the search for minimum risk paths. In Section 2.5 we explain this effect and also justify, why it is negligible in our case.

2.4 Experiments

We give an overview of the quality of individual models as well as their interplay in Section 2.4.1. Results of actual suggestions for train paths are given in Section 2.4.2.

2.4.1 Regression Models

To demonstrate the quality of the models, we created between-station models for the Zofingen-Lucerne corridor in Switzerland, as listed in Table 2.1. The residual standard error (standard error of the regression) S_E, i.e., the standard deviation of the difference between predicted and actual delay, is less than 30 seconds for the majority of the models, and not more than 50 seconds for any model. Apart from some outliers, which one would expect, the residuals are very moderate. Against the background of complicated dependencies between trains in real-world operations, the results are very encouraging.

To get a better impression on how the residuals are typically distributed, see Figure 2.6. The regression model for Wauwil-Sursee in Figure 2.6(a) has a very good fit, which also mirrors the fact that it does not seem to be a "critical" station. On the other hand the in-station model for Olten in Figure 2.6(b) has a less good fit, which might come from the more complicated structure of delays in Olten. In both plots, outlying points are rather below than above the diagonal, which is exactly what one would expect from a delay prediction model: Some delays are simply unpredictable.

The two residual plots also help to see to what extent the standard assumptions of linear regression modeling are satisfied. From both plots one can see that the linearity assumption $\mathbb{E}(\varepsilon_{i,\tau,d}) = 0$ seems to hold. On the other hand, the constant variance assumption does not seem to hold, the residuals look heteroscedastic. As this does not influence the unbiasedness and consistency of the used least

from	to	S_E	DoF	residuals					Mult. r^2	Adj. r^2
				min	1Q	2Q	3Q	max		
ZF	BRIT	9.2	11903	-27.8	-5.1	-1.2	3.7	126.0	0.9931	0.9931
BRIT	DAG	12.6	11903	-61.5	-4.8	-1.3	3.5	295.9	0.9874	0.9874
DAG	NEB	8.2	11902	-57.5	-3.5	-0.8	2.3	219.2	0.9948	0.9948
NEB	WAU	6.2	11902	-33.8	-3.4	-0.5	2.7	177.2	0.9971	0.9971
WAU	SS	18.3	11898	-105.1	-8.5	-2.3	5.4	370.5	0.9778	0.9778
SS	SEM	27.8	14273	-108.0	-14.6	-2.3	11.4	708.9	0.9403	0.9402
SEM	RBG	21.1	14274	-141.3	-7.9	-2.2	3.5	1032.0	0.9686	0.9685
RBG	HUEB	25.2	14274	-102.1	-11.4	-0.8	8.3	382.9	0.9582	0.9581
HUEB	EBR	16.3	19964	-58.5	-7.7	-1.7	4.6	663.7	0.9820	0.9820
EBR	GTS	49.0	19966	-223.1	-27.4	-5.9	18.9	587.5	0.8342	0.8340
GTS	LZ	41.2	37761	-202.4	-22.5	-7.3	14.0	760.5	0.8769	0.8768

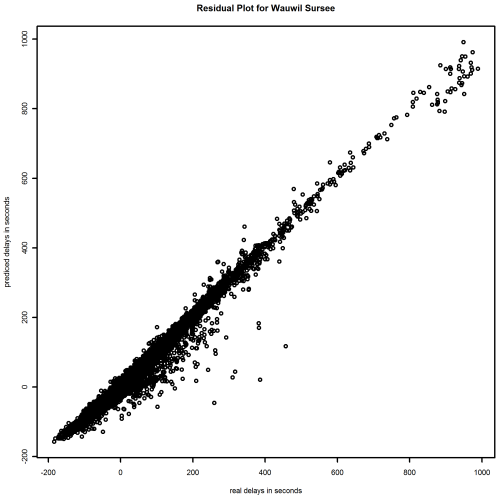
Table 2.1: Overview over the goodness of fit of the between-stations models on the Zofingen Lucerne corridor. The residual standard error is denoted by S_E, the degrees of freedom by DoF, the i-th quantile by iQ. Residuals and S_E are given in seconds.

squares estimators but rather the efficiency, this does not invalidate our approach: Given the very large amount of data that the models are estimated from, statistical efficiency is not our primary concern.

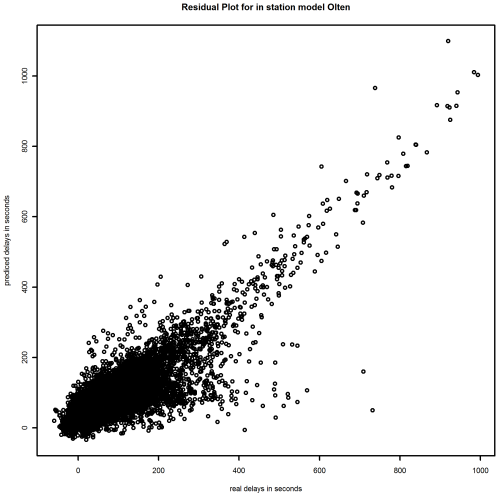
To demonstrate the possible quality of predicting a whole train path, we performed a cross validation on an extra train that drove on only 9 days, which were removed from the data before the models were learned. We then applied the models to the planned train path of the extra train. To see the best possible prediction, we set the initial delay at the first station to the actual delay of the extra train. The resulting predictions for each day are compared with the actual delays in Figure 2.7. Since the previous delay is an important predictor in all between-stations models, we also depicted the results for a "null" model, in which the previous delay is the only predictor used, in Figure 2.8. By comparison, one can clearly see that the additional predictors we selected have a significant impact on the quality of our models.

2.4.2 Shortest Path Algorithm

Continuing our example of the Zofingen-Lucerne corridor, we next give results for some hypothetical user requests. In Figure 2.9 all Pareto optimal train paths are depicted for the following user request: a fast train from Zofingen to Lucerne, earliest departure time 8:00, latest arrival time 9:30, no intermediate stops, maximum driving time of 130% (w.r.t. the minimum driving time). Interestingly enough, no Pareto optimal train path starts before 8:54. This means that all



(a) Wauwil-Sursee, between stations model



(b) Olten, in station model

Figure 2.6: Residual plots (a) for a between station model and (b) for an in station model

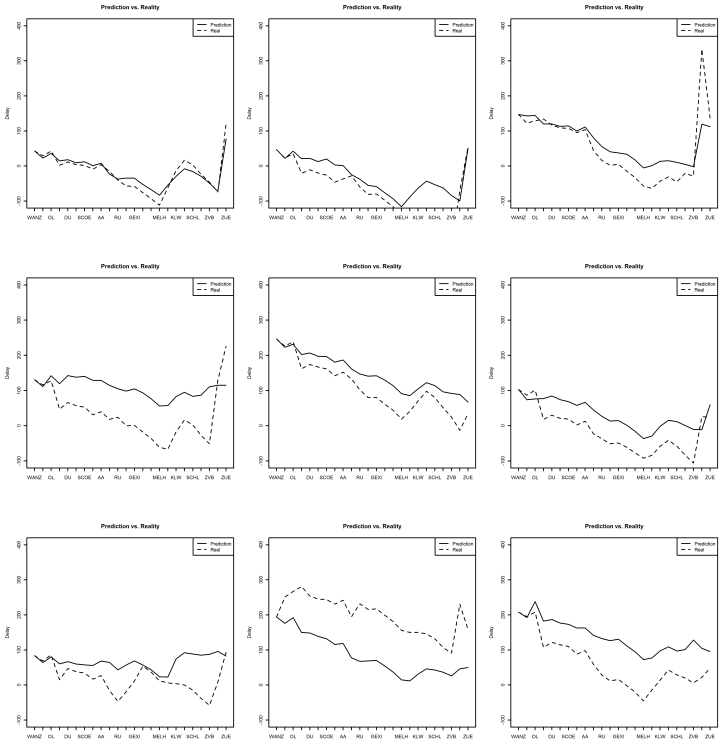


Figure 2.7: Delay profile and prediction for an additionally scheduled train, which drove on exactly the depicted nine days along the Zurich Bern corridor. The days where the train ran were taken out of the learning data.

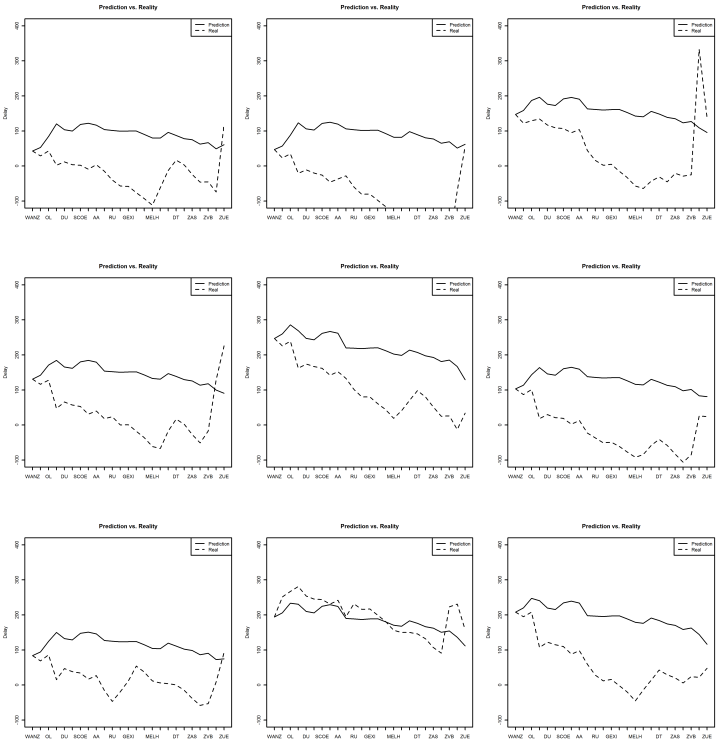


Figure 2.8: "Null" model, in which only the delay at the previous station is taken into consideration. One can see the significant improvement of the full model depicted in Figure 2.7 over these predictions.

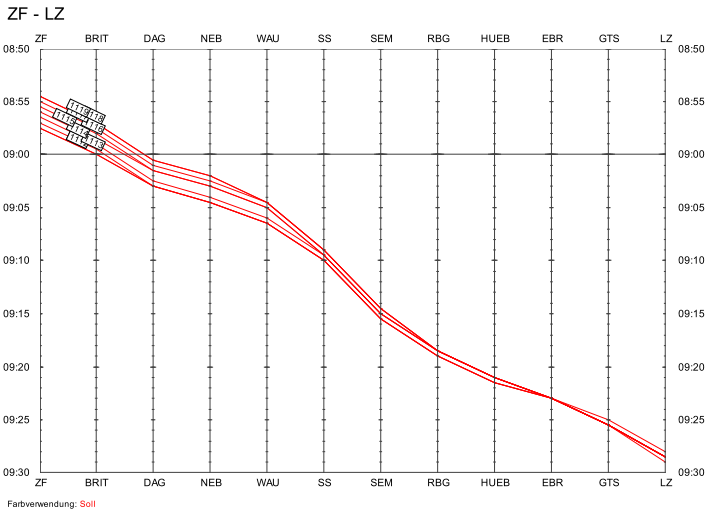


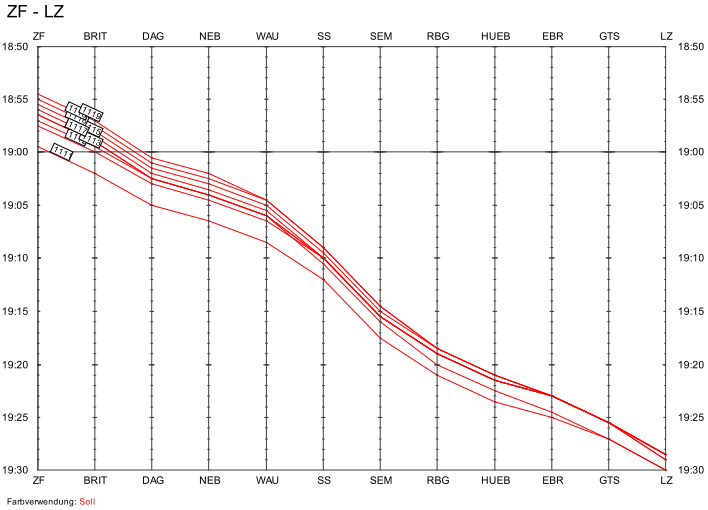
Figure 2.9: Suggested train paths for a request on the Zofingen-Lucerne corridor during the morning hours.

solutions departing before 8:54 are dominated by the final solutions.

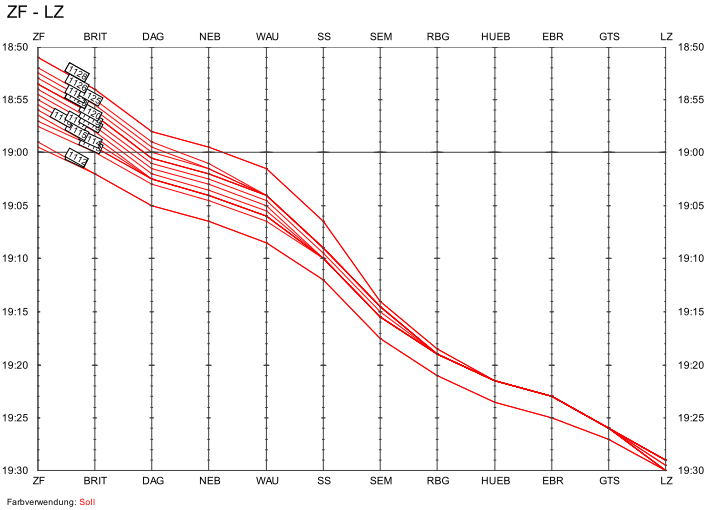
If one wants to increase the number of solutions, which is only possible if there are less solutions in the Pareto frontier than possilbe, c.f. Lemma 2.7, one parameter is the maximum driving time, given in percent of the minimum driving time. Consider the Figures 2.10(a) and (b). There, the corresponding user request is for a similar train path as above, with earliest departure time 17:00, and latest arrival time 20:00. In Figure 2.10(a), the maximum driving time was set to 130%, and in Figure 2.10(b) to 150%. Again, although the time windows are very large, the solutions give a clear indication where the train should be added.

2.5 Bias of Estimation

Even though the estimators provided by the linear regression models are unbiased, this unbiasedness is lost in the search for minimum risk paths. The reason for this is that in the shortest path algorithm the label at a node v_{i+1} for some fixed travel time t is set to the minimum



(a) Maximum driving time of 130%.



(b) Maximum driving time of 150%.

Figure 2.10: Suggested train paths for requests on the Zofingen-Lucerne corridor during the evening hours.

delay prediction with respect to a set $S^t_{v_{i+1}}$ of labels. The set $S^t_{v_{i+1}}$ contains for each of the ingoing edges (v_i, v_{i+1}) exactly one label corresponding to the delay prediction for the best path over this edge of time t. In layer 1 each label is a delay prediction from the linear regression model and therefore an unbiased estimator of the delay. Starting from layer 2, however, a label is computed as the minimum of delay-estimates of non-zero variance. It follows that the resulting delay estimates $\hat{\delta}$ have a negative bias, i.e., they underestimate the delay. What makes this bias delicate for the shortest path approach is that it depends both on the variance of the estimators and on the indegree of the vertices in G_r. To get an idea of the order of this bias, we have conducted the following experiment that mimics the minimization process: Note that for a fixed between-station model $\mathcal{M}_{i,i+1}$ (w.l.o.g.—the idea for an in-station model is similar) values of the indegree k are typically no more than 20, assuming a time granularity of 6 seconds between nodes. We draw k rows from the data, i.e., k delay events, uniformly at random and predict their delays $(\hat{\delta}^1, \ldots, \hat{\delta}^k)$. Each prediction is an unbiased estimate of the delay for the given row. We now compute the minimum $i = \arg\min_{1 \leq k' \leq k} \hat{\delta}^{k'}$ of these predictions and compare it to its real delay. The resulting difference $\hat{\delta}^i - \delta^i$ is the bias for this minimization over k predictions. By repeating this process many times and averaging we get estimates for the bias for this value of k on model $\mathcal{M}_{i,i+1}$. On our instances the variations in these estimates for realistic indegrees are completely dominated by the residual standard errors of the estimation.

With such indegree-dependent estimations at hand it is possible to integrate the bias into the shortest path computation by simply charging these extra costs to the edges, depending on the indegree of the nodes. We did not implement this, partly because the residual standard error seems to dominate the bias, partly because the above estimation process ignores the complicated correlation between the predictions over which the minimum is taken. For this reason, the above estimation process is probably not completely unbiased by itself. Still it should give a good idea about the relative (in-)significance of the effect.

2.6 Conclusion

In this chapter, we presented a novel approach that helps planners with adding further train paths to a corridor. The approach profits from extensive delay data recorded by SBB and keeps further infrastructure modeling efforts to a minimum. We proposed a combination of linear regression models and a shortest path algorithm that yields a Pareto front of suggestions with respect to the trade-off between risk and travel time.

Currently, searching for an additional train path in an already dense corridor is a laborious task. For long corridors, the task might be split between several planners who look for ways to rearrange the trains in the stations and reroute them on the tracks. At SBB time consuming simulations are carried out for critical situations.

Our model is intentionally based only on the most important feasibility requirements. The set of solutions should be taken as recommendations to the planner who can then select the most promising ones and then refine and tune these. Our approach is both much faster and less labor-intensive than a detailed simulation, and yet allows for very reasonable risk estimates. We believe that our approach can reduce the time planners need to find the train path with best quality.

A long term goal would be to integrate this approach into existing planning tools at SBB. For this goal, one could improve the current approach in three directions: First, it would be interesting to modify the risk measure in such a way that it also directly considers the delay of follow-up trains caused by the additional train. From the results of Section 2.3.2 it seems difficult to find one that keeps the problem polynomial time solvable. Second, our experiments indicate that on very dense corridors the operational constraints, as modeled in our approach, make the addition of a train infeasible. Therefore, one could evaluate an approach that removes (few) trains that partially block the corridor and can be rescheduled later so that a (more critical) train that goes along the full corridor can be added. Finally, an even more flexible approach would also allow to adjust the train paths of the already planned trains of the corridor. Such a modification, however, has to guard against problems of infeasibility with further constraints which are currently not modeled.

Chapter 3

Mining for Dependencies in Delay Data

3.1 Introduction

During operation trains can get delayed for various reasons: customers blocking doors, train connections, scarce track capacities, bad weather conditions, technical problems, etc. From a planner's point of view, some causes for delay just have to be accepted, such as customer behavior, and some have to be dealt with in disruption management, such as power failure due to catastrophic weather conditions. There are, however, also systematic dependencies between the delays of trains, which are inherent to the timetable and can be influenced by careful planning. In this chapter we present algorithmic methods to efficiently detect such dependencies in large-scale, real-world railway delay data. The goal is to support planners in improving timetables by providing them with a list of potentially systematic delay dependencies of the current timetable. These dependencies can then be more closely examined by appropriate statistical methods in a following step. Finally, planners may be able to remove or weaken those dependencies by means of small, local modifications to the timetable.

At Swiss Federal Railways (SBB), delay data are obtained by the interlocking system throughout the whole Swiss railway network and recorded on a less detailed level comprising about 2300 operating

points. These data contain the arrival and departure times of each train for every operating point along its route for every day of operation. There are, however, no data for the dependencies between delays of different trains.

Delays are usually classified into primary and secondary delays. Primary delays "occur" at some point in the network, e.g., due to doors blocked by customers, technical problems, or accidents. Secondary delays (also called secondary delays) are the consequences of primary or secondary delays of other trains. For example, a punctual train may accumulate a secondary delay because it waits for a delayed train to maintain a connection. Another example is a pair of trains that need to leave a station via the same track segment in a fixed order, where the first train leaving the station is late, forcing the second train to wait until the track segment is free. If the delay of a train causes a secondary delay of another train on a regular basis, e.g., on at least 25% of the days, we speak of a *systematic dependency* between the delays.

In this chapter, we suggest models that, given certain parameters, describe the patterns underlying the most important types of dependencies. We present algorithms that efficiently find systematic dependencies in large-scale railway delay data. If a train depends on the delays of several other trains, the most significant dependency for the delay of each day can be identified by our methods. Our approach does not rely on any assumption on the statistical distribution of the data. We show results of our method on real-world data.

The chapter is organized as follows. In Section 3.1.1, we give a brief summary of related work. Section 3.2 introduces the models of dependencies along with the algorithms to detect them. We show how the delays of a single train can be explained by several dependencies in Section 3.3. In Section 3.4, we suggest modifications of the algorithms to account for errors in the data or exceptions to the model. Finally, we present results of our experiments in Section 3.5, and give a conclusion and outlook in Section 3.6.

3.1.1 Related Work

In her PhD thesis [14], Conte examines several approaches to identify dependencies among delays. Arrival and departure delays of trains are associated with random variables. Assuming a multivariate nor-

mal distribution, the Tri-graph method [85] is applied to construct a graph whose nodes represent the random variables. In such a graph, edges are included on the basis of non-zero (partial) correlation coefficients, hence missing edges represent conditional independence. Conte and Schöbel [15] suggest to use the constructed Tri-graph in combination with linear regression to generate so called virtual constraints for the delay management problem. For the latter, refer to, e.g., [43, 75].

In this chapter, we present an algorithmic approach that makes no assumptions about the distribution of delays. Furthermore, we give real-world examples of dependencies that have very low correlation coefficients, and yet are important. In contrast to the network-wide approach suggested by Conte, however, we are currently detecting dependencies only within a station. Further, our goal is to support planners in improving timetables, rather than making robust delay management decisions during operations.

The problem of distinguishing between primary and secondary delays is not only of interest for timetabling, but also for determining fines due to performance contracts between governments and train operating companies. Daamen, Goverde and Hansen [17] developed a prototype software to register secondary delays due to conflicts on track sections. Their approach requires detailed delay data at the level of signals and track segments. Further, the approach requires dispatchers to identify incidents leading to primary delay. The detection of secondary delays due to waiting for a connection would be possible in their approach given that scheduled connection times are provided.

In contrast, our approach aims at finding systematic dependencies in the timetable rather than precisely matching particular delays to train operators. Our approach works with less detailed data on the level of operating points, requires no incidence records, and recognizes both dependencies due to maintaining connections and due to blocked track sections.

For an overview of other delay propagation models, including those interested in prediction, refer to [14, 21, 8, 87].

3.1.2 Summary of Results

We suggest a novel approach to detect dependencies in delay data, allowing planners to identify trains that suffer from secondary delays

in a systematic fashion. We present efficient algorithms to detect two of the most important types of dependencies, namely dependencies due to resource conflicts and due to maintained connections. We give experimental results on real-world delay data that demonstrate the practical applicability of our algorithms.

3.2 Models and Algorithms

Two important types of dependencies between delays of different trains are the waiting dependency and the blocking dependency. In this section, we formally characterize such dependencies between a train that is originally delayed, called the *source*, and the train to which the delay propagates, called the *victim*. To be more precise, when we speak of a delayed train, we actually mean that some event, i.e., the arrival or departure of a train at a specific station, occurs later than scheduled.

In the following, we denote by τ_x a delayed train, called the *source*, that (potentially) causes secondary delay of another train τ_y, called the *victim*. Further, we denote by x_d and y_d the delay (i.e., the difference of the actual and planned time of a certain event) of the (potential) source and victim trains, respectively, on day d.

3.2.1 Waiting Dependency

A *waiting dependency* is given if a so called *connecting train* waits for passengers of a *feeder train* in order to maintain a connection. Hence, the delay of the arrival event of the feeder train may propagate to the departure event of the connecting train at a specific station. In order to find such dependencies in the data, we first formulate an idealized model of a waiting dependency. We remark that models of this kind are already known, e.g., see [48]. Based on this model, we provide an algorithm that finds waiting dependencies in the data.

Ignoring for a moment that the victim may depend on more than one source and may also suffer from other sources of delay, we can model an idealized waiting dependency as follows. First, there usually is some buffer time b up to which the feeder train may be delayed without affecting the connecting train. If the feeder train is delayed

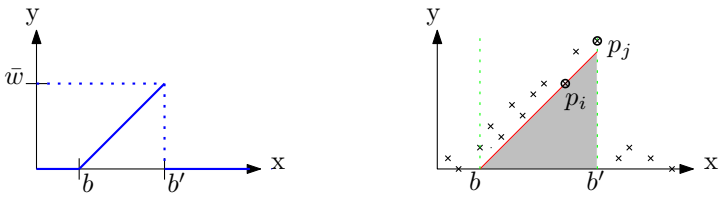


Figure 3.1: (a) Idealized waiting dependency of the delay of a connecting train y on the delay of a feeder train x within the interval $[b, b']$. The maximum waiting time is $w = b' - b$. (b) Hypothetical example data, where each point $p_d = (x_d, y_d)$ corresponds to the observed delays on day d. The interval $[b, b'] = [x_i - y_i, x_j]$ is the solution of Problem 3.2, i.e., it maximizes the number of points $|S|$ above the shaded triangle, subject to the condition that no point may lie within the triangle.

by more than b, the connecting train will wait to maintain the connection, but only up to a maximal waiting time $\bar{w}$, which corresponds to a delay $b' = b + \bar{w}$ of the feeder. Denoting by x_d the delay of the feeder train on day d, and by $\tilde{y}_d$ the (idealized) corresponding delay of the connecting train, the waiting dependency can be formulated as

$$\tilde{y}_d = f(x_d, b, b') = \begin{cases} x_d - b & b \leq x_d \leq b' \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

as shown in Figure 3.1(a). The parameters b and b' may vary depending on the station and the specific pair of trains involved in the scheduled connection. We remark that planned values for these parameters could be obtained in principle. Unfortunately, it may turn out that during operations, the actual parameters differ from the planned ones. We therefore have to assume that the actual parameters are unknown.

In practice, of course, one victim train may depend on several source trains, and furthermore, there may be other causes of delay. Therefore, $f(x_d, b, b')$ can only be a lower bound on the actual victim's delay within the interval $[b, b']$. As we are interested in systematic dependencies where delay is propagated on a regular basis, we want to find an interval $[b, b']$ containing a maximum number of

points $p_d := (x_d, y_d)$, for which $f(x_d, b, b')$ is a lower bound on the delay y_d of the victim. Formally, given the delay data x_d and y_d for a set of days $d \in D$ for potential source τ_x and victim τ_y, respectively, we get the following problem:

$$\max_{b,b'} |S| \tag{3.2}$$
$$\text{s.t.} \quad S = \{(x_d, y_d) \mid d \in D, b \leq x_d \leq b',\ y_d \geq x_d - b\}$$
$$\emptyset = \{(x_d, y_d) \mid d \in D, b \leq x_d \leq b',\ y_d < x_d - b\}$$

Geometrically, we are looking for a rectangular triangle with a maximum number of points above it but none within. An example is given in Figure 3.1(b). The problem is solved by Algorithm 2, which works as follows: Each point p_d, $d \in D$ implicitly defines an interval on the x-axis, namely $[x_d - y_d, x_d]$, that is a potential solution for Problem 3.2. The algorithm sweeps through the points p_d, $d \in D$, in non-decreasing order of x_d. Now, a current solution $[x_d - y_d, x_d]$ can be extended to the next point if it lies above the 45 degree line through point p_d. Otherwise, the current solution contains a maximal number of points, and the next point is the first point of another solution. Keeping track of $|S|$ of Problem 3.2 is straightforward, since the start of the corresponding next interval is to the right of the start of the previous interval.

Theorem 3.1. *Algorithm 2 computes a solution to Problem (3.2) in time* $\mathcal{O}(n \log n)$.

Proof. Every point $p_i = (x_i, y_i)$ defines an interval $[b_i, b'_i]$ and a corresponding set of points S as follows: The interval starts at the intercept of the 45 degree line through p_i with the x-axis, namely at $b_i := x_i - y_i$. The interval ends at $b'_i = x_j$, the x-coordinate of the rightmost point p_j of the sequence of points above the line, i.e., for all p_k, $k \in \{i, \ldots, j\}$ it holds that $y_k \geq x_k - b_i$.

Notice that in order to maximize $|S|$ it suffices to examine only those intervals $[b_i, b'_i]$ which are defined by the points p_i with $i \in \{1, \ldots, n\}$: In any optimal solution (S^*, b^*, b'^*) there exists one point $p_i^* \in S^*$ with maximal intercept b_i^*. Hence, the start b^* of the optimal interval must be greater or equal to b_i^*, for otherwise p_i^* would not be in S^*. So setting $b^* = b_i^*$ is feasible for all points in S^*, as well as setting the end of the interval $b'^* = x_j^*$, with p_j^* being the rightmost point of S^*.

Algorithm 2: Detect Waiting Dependency

Input: Delays $p_d = (x_d, y_d)$ of source τ_x and victim τ_y on days $d \in \{1, \ldots, n\}$.
Output: Number of points k^* in optimal interval $[b^*, b'^*]$

Sort data according to non-decreasing x_i, breaking ties according to non-increasing y_i ;
$p_{n+1} \leftarrow (\infty, 0)$; `// sentinel`
for $i \leftarrow 1$ **to** $n+1$ **do**
  $b_i \leftarrow x_i - y_i$; `// calculate intercepts`
end
$k, k^* \leftarrow 0$; `// number of points in current / best solution`
$b \leftarrow b_1; b^* \leftarrow 0$; `// start of current / best solution`
$\ell \leftarrow 1$; `// index of leftmost point in current solution`
for $i \leftarrow 1$ **to** $n+1$ **do**
  if $b_i > b$ **then**
    `// cannot extend current solution to` p_i
    if $k > k^*$ **then**
      `// update best solution`
      $k^* \leftarrow k$;
      $b^* \leftarrow b$;
      $b'^* \leftarrow x_i - 1$;
    end
    `// initialize new solution`
    $b \leftarrow b_i$;
    `// find first point in new interval`
    while $x_\ell < b$ **do**
      $\ell \leftarrow \ell + 1$;
    end
    $k \leftarrow i - \ell + 1$;
  else
    $k \leftarrow k + 1$;
  end
end

The algorithm sweeps through all points in the order defined on Line 1 of Algorithm 2. Maintaining b as the starting point of the current interval, it maximally extents the interval until the first point below the 45 degree line is met, i.e., the condition on Line 10 is violated. The intervals corresponding to the points above the 45 degree line need not be considered, since they either are infeasible or contain only a subset of the points of the current interval.

Clearly, sorting takes $\mathcal{O}(n \log n)$ time, and the rest of the algorithm runs in time $\mathcal{O}(n)$. □

To detect all waiting dependencies in the data, Algorithm 2 is run on data of pairs of trains that are scheduled to meet at station within a reasonable time difference, say, up to 15 minutes. Depending on the number of days recorded in the data, we define a minimum number of days that must be in S^* in order to call a dependency *systematic*. For the points in S^* of a systematic dependency, we say that the delay of the victim is *explained* by the dependency, meaning that the delay of the source minus the buffer time b^* is a lower bound on the delay of the victim on the days corresponding to the points in S^*.

3.2.2 Blocking Dependency

If two trains have to use the same infrastructure element, such as a track segment or a platform, then a dependency between their delays may exist, since one of them must pass that element first. We call such a dependency a *blocking dependency*. This dependency can occur between any combination of arrival and departure events, as exemplified in Figure 3.2. For reasons of operational safety, a certain headway time must be respected between two consecutive trains accessing the same infrastructure element. If we depict the delay data of two blocking trains as in Figure 3.3, one can identify a 45 degree line representing all the hypothetical arrival/departure times that would lead to a crash of the two trains. In our model of a blocking dependency, we assume that certain known, minimal headway times are always respected. Hence, there is a stripe around the 45 degree line in which no points may lie. We require that the stripe has a minimum width $\underline{w}$, which follows from the given headway times. The stripe also partitions the points according to the precedence of the trains. In some cases, such as in Figure 3.2(b), it may practically not be possible to switch the order of trains, even in case of large delays, such that

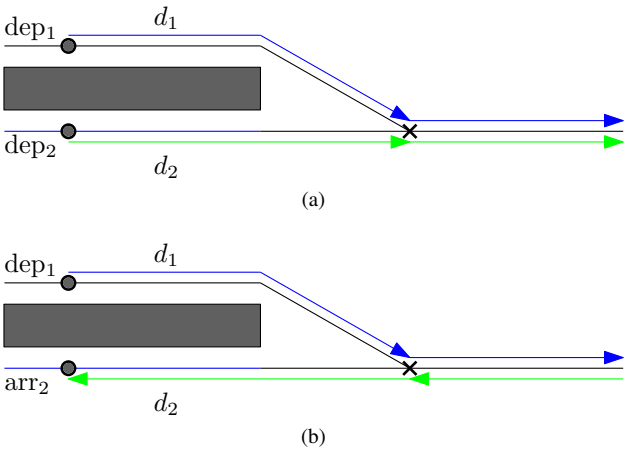


Figure 3.2: Examples for conflicting trains, driving (a) in the same direction and (b) in opposite directions. From their arrival/departure location, they have to travel distances d_1 and d_2 to their first point of conflict.

one region is empty. Note that if the order of trains is fixed, a blocking dependency would also be found by Algorithm 2. We remark that the order of trains on the conflicting track segment is not obvious from the data, since delays are not given at the level of track segments but at the more aggregated level of operating points. Therefore, data about the exact routes of the trains is not available.

As in the case of waiting dependencies, we are searching for a subset of points for which a function of the delay of the source is a lower bound on the delay of the victim. Hence, we are interested in all points above the stripe. Formally, given the delay data x_d and y_d for a set of days $d \in D$ for potential source τ_x and victim τ_y, respectively, we get the following problem:

$$\max_{b,b'} |S| \tag{3.3}$$

$$\begin{aligned} \text{s.t.} \quad & S = \{(x_d, y_d) \mid d \in D, x_d \geq b,\ y_d \geq x_d - b\} \\ & \emptyset = \{(x_d, y_d) \mid d \in D, x_d - b' < y_d < x_d - b\} \\ & b \leq b' - \underline{\mathrm{w}} \end{aligned}$$

A subtlety of blocking dependencies is that there may be an inter-

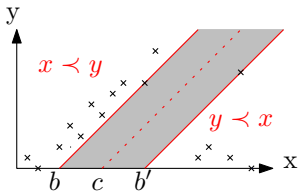


Figure 3.3: Example of a blocking relationship between two trains departing in the same direction. The axes denote the delay of the respective train. The intersect c defines a 45 degree line (dotted) on which each point would represent a crash scenario, so $c = (\text{dep}_y + d_y) - (\text{dep}_x + d_x)$, where dep_i is the planned departure time of the event and d_i the driving time to the conflict point of train τ_i, as shown in Figure 3.2. Around this line, a stripe (solid lines) represents the headway time that has to be respected, ensuring a safety distance between the trains. Every point above the stripe represents a situation where train τ_x precedes train τ_y on the track segment of conflict, and vice versa for points below the stripe.

val in which there are points both above and below the stripe. Therefore, the lower bound on the delay of the victim is defined for a subset S of points, rather than for all points falling in an interval. Algorithm 3 solves Problem (3.3) using a sweep-line approach: Observe that if a blocking dependency exists, the sought after stripe (of width at least $\underline{\text{w}}$) in which no point lies is delimited by two 45 degree lines, each going through one of the given points p_d, $d \in D$ (assuming $|D| \geq 2$). These two points must be consecutive if we order the points according to the intersects of their corresponding 45 degree lines with the x-axis. Furthermore, in such an ordering all points lying above the sought stripe will be encountered before the first point below the stripe is encountered.

Theorem 3.2. *Algorithm 3 computes a solution to Problem (3.3) in time* $\mathcal{O}(n \log n)$.

Proof. Every point $p_i = (x_i, y_i)$ defines a 45 degree line through the intercept $(b_i, 0)$ and itself. W.l.o.g., we consider only those stripes whose left and right intercepts are defined by consecutive intercepts of the points (and hence, there are no points in the stripe). The condition on Line 14 of Algorithm 3 ensures that only stripes respecting

Algorithm 3: Detect Blocking Dependency

Input: Delays $p_d = (x_d, y_d)$ of source τ_x and victim τ_y on days $d \in \{1, \ldots, n\}$. Minimum width of stripe $\underline{w} > 0$.
Output: Number of points k^* above the optimal stripe defined by b^* and b'^*.

```
1  Sort data according to non-decreasing x_i;
2  p_{n+1} ← (∞, 0) ;                                  // sentinel
3  for i ← 1 to n + 1 do
4  |  b_i ← x_i − y_i ;                  // calculate intercepts
5  end
6  c ← array of sorted intercepts {b_1, ..., b_{n+1}};
7  k, k* ← 0 ;                          // number of points in
   current/best solution
8  b, b* ← 0 ;                  // left intercept of solution
9  b', b'* ← 0 ;               // right intercept of solution
10 ℓ ← 1 ; // index of first point above stripe
11 for j ← 2 to n + 1 do
12 |  b ← c[j − 1] ;
13 |  b' ← c[j] ;
14 |  if b' − b ≥ w then
15 |  |  while x_ℓ < b do
16 |  |  |  ℓ ← ℓ + 1 ;          // p_ℓ is leftmost point
   |  |  |  above stripe defined by b and b'
17 |  |  end
18 |  |  k ← j − ℓ ;              // number of points above
   |  |  stripe
19 |  |  if k > k* then
   |  |  |  // update best solution
20 |  |  |  k* ← k;
21 |  |  |  b* ← b;
22 |  |  |  b'* ← b';
23 |  |  end
24 |  end
25 end
```

the minimum width $\underline{w} > 0$ are considered. When k is computed on Line 18, it holds that $b' > b$ and that the points below the stripe defined by b and b' all have an intercept greater or equal to b'. Thus, there are $n-(j-1)$ points below the stripe (assuming there are non-negative delays only), and there are $\ell - 1$ points to the left of the intercept b of the stripe. Hence, the number of points above the stipe is $k = n - (n - j + 1) - (\ell - 1)$. It follows that the algorithm correctly computes a solution to Problem (3.3). Clearly, sorting takes $\mathcal{O}(n \log n)$ time, and the rest of the algorithm runs in time $\mathcal{O}(n)$. □

As for the waiting dependency, we consider only those dependencies to be systematic which hold on at least the minimum number of days required. To account for reasonable headway times, we further require an appropriate minimum width of the stripes. The detection of pathological cases can be prevented by computing reasonable bounds on the location of the center of the stripe from the timetable.

3.3 Multiple Dependencies

It is straightforward to generalize the lower bound obtained from a single dependency to the case where a train is the victim of several dependencies. In the following, we assume that for a victim train several such dependencies have been found. Thus, we may get several lower bounds on the delay of the victim on a particular day, namely from those dependencies that can explain it on that day. We make the usual assumption that the victim is delayed by the worst cause, i.e., the source providing the maximum lower bound for the victim's delay.

Formally, we are given a train τ_y that is the victim of k dependencies with sources τ_x^i, $i \in \{1, \ldots, k\}$. Generalizing from the lower bound for the victim's delay from above, let us define

$$g\left(f_1(x_d^1), \ldots, f_k(x_d^k)\right) = \max\left\{f_1(x_d^1), \ldots, f_k(x_d^k)\right\}, \qquad (3.4)$$

where f_i is the function that maps the delay of source τ_x^i to a lower bound of the delay for the victim, as determined by the corresponding waiting or blocking dependency, i.e.,

$$f_i(x_d^i) = \begin{cases} x_d^i - b^i & \text{if } (x_d, y_d) \in S^i, \\ 0 & \text{otherwise} \end{cases}, \qquad (3.5)$$

where S^i and b^i are the solution of Problem (3.2) or Problem (3.3) for the particular waiting or blocking dependency, respectively.

It follows that for each day $d \in D$, the function g is a lower bound on the delay y_d of victim τ_y on that day. We call the source τ_x^i for which $g\left(f_1(x_d^1), \ldots, f_k(x_d^k)\right) = f_i(x_d^i)$ the *best explanation* for y_d. Note that for a given day d, there may be no source explaining y_d, yielding only a trivial lower bound as the best explanation.

An example of such a multiple dependency is given in Figures 3.5, 3.6, and 3.7, where we plot the victim's delay against the best explanation for each day.

3.4 Extensions

The recorded delay data are subject to inaccuracies, because the measurements on the tracks are aggregated to the level of operating points. SBB requires from their systems that such errors in the data be less than 20 seconds. Furthermore, it may well be the case that on a few days, the operational waiting rule described by Model (3.1) is violated. Such exceptions are unavoidable during operations. They may be caused by human mistake or as an intentional reaction to an exceptional situation.

For these reasons, there may be points p_i in the data that one would like to ignore, because otherwise, they may prevent a dependency from being detected. A similar problem in statistics is known as the least trimmed squares estimator for linear regression as surveyed in [52], where one seeks to find a subset of points minimizing the squared residuals for that subset. In our case, however, we are restricted to subsets corresponding to intervals, and have a fixed slope for the line we would like to "fit".

Exceptional points may prevent Algorithms 2 and 3 from detecting a dependency completely or worsen the resulting lower bounds (by increasing b). It is possible to extend the algorithms to allow for a maximal number $\overline{r}$ of allowed exceptional points. Clearly, practical values of $\overline{r}$ are very small. In the case of waiting dependencies, we

want to solve the problem

$$\max_{b,b'} |S| \tag{3.6}$$
$$\text{s.t.} \quad S = \{(x_d, y_d) \mid d \in D, b \leq x_d \leq b',\ y_d \geq f(x_d, b, b')\}$$
$$\overline{r} \geq |\{(x_d, y_d) \mid d \in D, b \leq x_d \leq b',\ y_d < f(x_d, b, b')\}|$$

We sketch the necessary modifications of Algorithm 2 in order to solve Problem (3.6): We introduce a variable r that keeps track of the number of exceptional points in the current solution's interval $[b, b']$. Further, we keep a priority queue of these r points ordering them by their intersects b_i. We need the queue because points will leave the solution in the order of their intersects, whereas they enter the solution in the order of their x-coordinate. Now, we modify the criterion in Line 10, such that a solution is only extended if less than $\overline{r}$ exceptional points are in the current solution. If the solution is extended and the current point is exceptional, we add it to the priority queue. If the solution is not extended, i.e., there are already $\overline{r}$ exceptional points in the current solution, we remove the point p_l with smallest intercept s_l from the queue and increase b to s_l.

During execution of the algorithm, no more than $\overline{r} < n$ points are in the priority queue, and each point may only be inserted and removed once, at a cost of $\mathcal{O}(\log \overline{r})$. Hence, Problem 3.6 can still be solved in time $\mathcal{O}(n \log n)$.

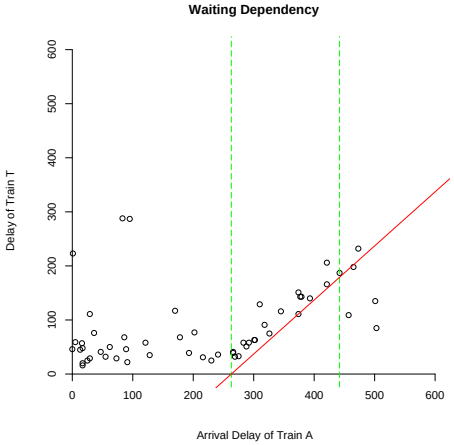
3.5 Experiments

In this section, we present some of the dependencies which can be found in real-world data. The data comprised several important operating points of the SBB network during two months of the 2008 timetable. We required a minimum number of 15 explained delays for all dependencies, a minimum interval width of 90 seconds for waiting dependencies, as well as a minimum interval width of 120 seconds for blocking dependencies. The following plots were created with R [83], as well as the correlation statistics (based on Pearson's product moment correlation coefficient).

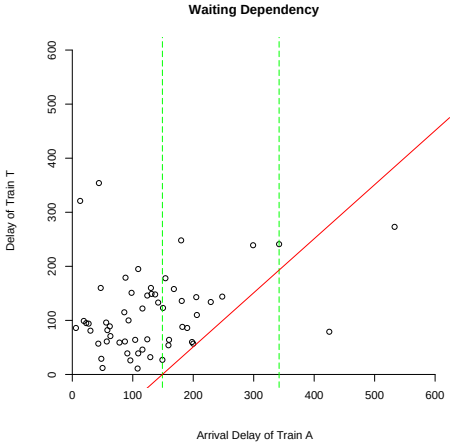
The dependencies in Figures 3.4 and 3.5 are single waiting and blocking dependencies, respectively, for different victims. Some of the examples presented here were specifically selected to demonstrate

that there are important dependencies which have a rather low Pearson's correlation coefficient. Given that we expect that only a subset of the points are part of the dependency, this is not surprising. However, one can also find a low correlation when looking only at the explained points, e.g., those which lie in the interval of a waiting dependency, see Figure 3.4(b). With many of the single dependencies we found, it is problematic to asses how severe the influence of a source train is on a victim train, since there may be other sources of delay (e.g., customers blocking doors), as well as measurement errors in the data, that may prevent us from finding the correct parameters b, b'. Therefore, one still relies on a planner with knowledge of the timetable and the particular station in order to asses whether there is a causal relationship between source and victim.

The picture becomes clearer once several trains are sources of secondary delays for a victim train. Examples of multiple dependencies are given in Figures 3.5, 3.6, and 3.7, each showing a victim for which several dependencies could be found. We included a plot of the victim's own arrival delay, where available, and plots showing the best explanation for its delay upon departure for each day. Notice that in the examples of Figures 3.5 and 3.6, arrival and departure delays of the victim do not follow an obvious pattern. Looking at the best explanation plot, however, there is an almost linear dependency of the departure delay of the victim on the respective best source. (Notice that a perfect explanation would have all points on a 45 degree line through the origin.) In these cases, it seems clear that most of the delay of the victim is due to secondary delays.

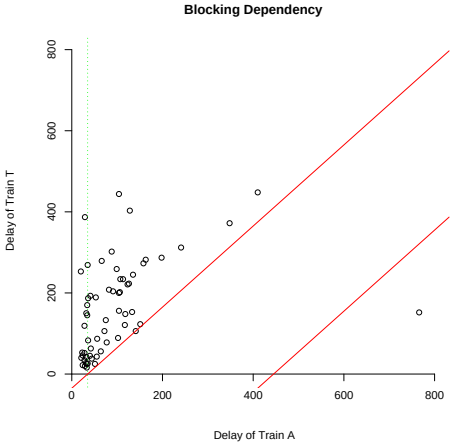


(a)

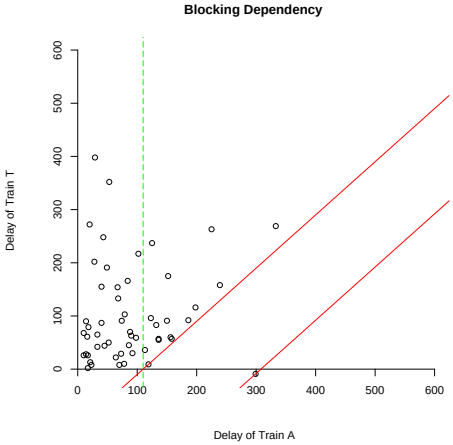


(b)

Figure 3.4: Two examples of waiting dependencies in Basel. (a) The correlation between arrival and departure over all days is as low as 0.1602 (with a p-value of 0.2215). In the explained interval (between the dashed lines), the correlation is 0.9513 (p-value 3.652e-11). (b) In this example, the correlation over all days is 0.2151 (p-value 0.0959), higher than the correlation over the explained interval, which is 0.0998 (p-value 0.6845), due to an event of train T with a delay above 10 minutes.

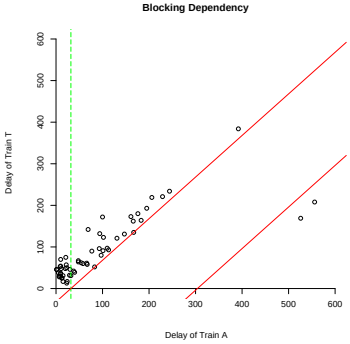


(a)

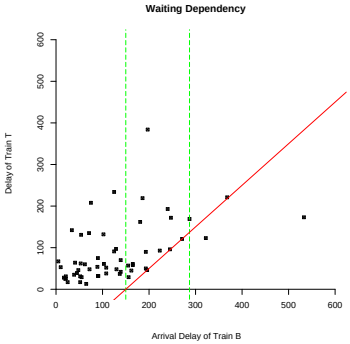


(b)

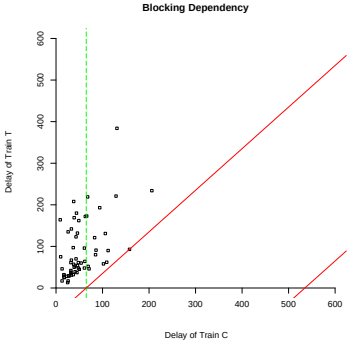
Figure 3.5: Examples of blocking dependencies. (a) Two departures in Bern, blocking each other; the correlation is 0.3969 (p-value 0.0015). (b) Blocking dependency in Basel, with correlation 0.2040 (p-value 0.1147).


Blocking Dependency
Delay of Train T
Delay of Train A
0
100
200
300
400
500
600

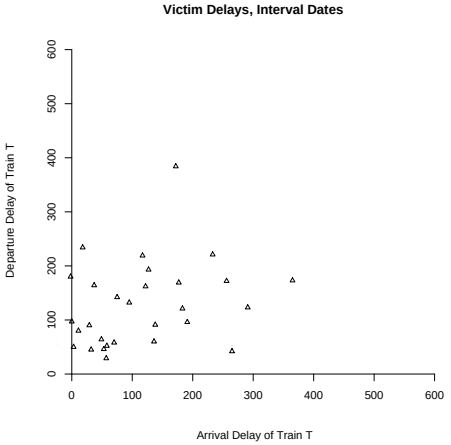
(a)


Waiting Dependency
Delay of Train T
Arrival Delay of Train B
0
100
200
300
400
500
600

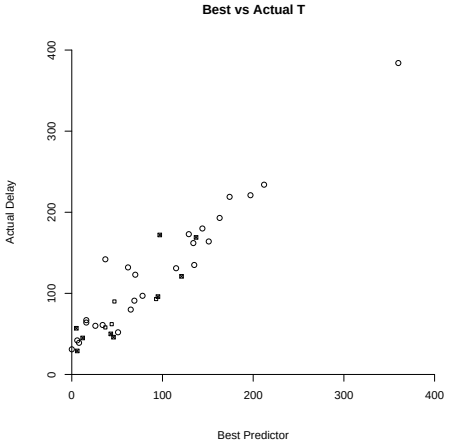
(b)


Blocking Dependency
Delay of Train T
Delay of Train C
0
100
200
300
400
500
600

(c)



(d)



(e)

Figure 3.5: Multiple dependency of a victim train T with sources A, B, and C. (a) source A blocking T on 34 explained days. (b) Waiting dependency with B, 17 days explained. (c) source C blocking T on 16 days. (d) The arrival and departure delays of T have no obvious pattern. (e) Best explanation by sources. A (circles), B (filled squares), and C (hollow squares) is the best explanation on 65.8%, 23.7%, and 10.5% of the explained days, respectively.

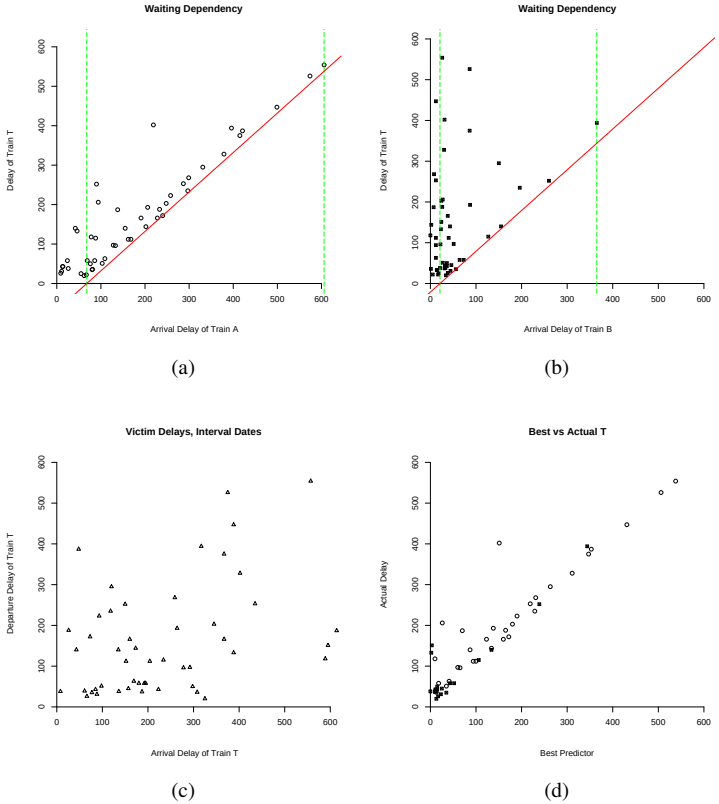


Figure 3.6: Multiple dependency of a victim train T with waiting dependencies only. (a),(b) Waiting dependencies with trains A, B, on 17 days, each. (c) The arrival and departure delays of T have no obvious pattern. (d) Best explanation by sources. A (circles) and B (filled squares) are the best explanation on 60.8% and 39.2% of the explained days, respectively.

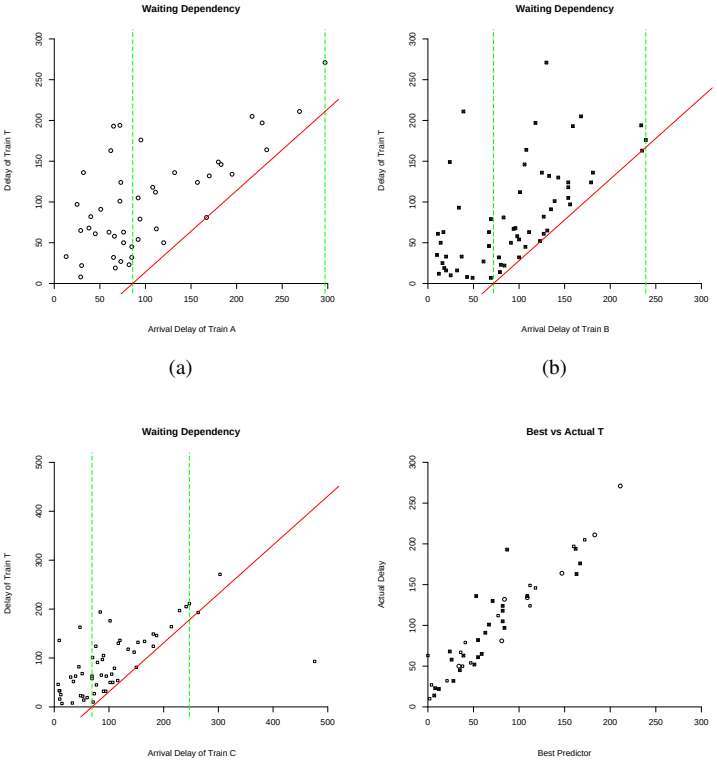


Figure 3.7: Multiple dependency of a train T on three trains at the first station of T's trip, for which no arrival delay is available in the data. (a),(b),(c) Waiting dependencies with trains A, B, and C, explaining 20, 38, and 35 days, respectively. (d) Best explanation: A (circles), B (filled squares), and C (hollow squares) are the best explanation on 15.2%, 54.3%, and 30.4% of the explained days, respectively.

3.6 Conclusion

The results of our experiments are very encouraging. We analyzed real-world delay data from SBB. Using the approach presented in this chapter, we were able to find systematic dependencies which have significant impact on daily operations. We implemented a prototype that connects to the SBB database which planners may use to analyze the delays at all operating points of the Swiss railway network.

Our approach is useful to quickly find candidate dependencies from large real-world data sets that provide lower bounds on the delay of trains. Especially in the case of multiple dependencies, we are able to find examples indicating that most of the delay of a train is due to secondary delays. Allowing for exceptional points as described in Section 3.4 turned out to be helpful. A drawback of our approach is that it does not provide a measure of statistical significance of the found candidate dependencies. Therefore, we still rely on the knowledge of planners in order to validate that the observed delays are actually a problem of the timetable.

Therefore, in a second step, a statistical examination of the dependencies could be useful to assess their significance, especially in the presence of exceptional data points. It would be interesting to collect additional data that allows the distinction between primary and secondary delay, and to compare these with the dependencies found by our algorithms.

Finally, it would be interesting to extend our approach to global dependencies, i.e., to trace back the propagation of delays along the route of trains, possibly yielding a network of delay propagations. Ideally, it would be possible to estimate the effect of a small local change of the timetable, say, by adding a small buffer time, on a network of delay propagations.

Part II

Optimizing Operations at Classification Yards

Chapter 4

Track Allocation at Classification Yards

4.1 Introduction

A rail-freight transportation network is used to transport goods between two points in the network. Efficient delivery of goods in a rail-freight transportation network requires a careful planning at many different levels, such as assigning cars to trains, routing trains through the network, and scheduling the cargo trains together with passenger trains that share common track infrastructure.

One distinguishes between full train load and single wagon load traffic. In full train load transportation, all wagons of a train belong to the same customer and share the same origin and destination. Typical examples are trains transporting coal or ore. In single wagon load traffic, a freight train consists of cars with various origins and destinations, often from different customers. Typically, these customers are from the midsized industries.

The problems we study in this chapter arise from single wagon load traffic. In order to transport each car to its destination, the freight trains are disassembled into individual cars and formed into new outbound trains at certain terminals in the network. This process is called *classification* (also *marshalling* or *shunting*), and is conducted in so called *classification yards* (also known in the literature as *marshalling*

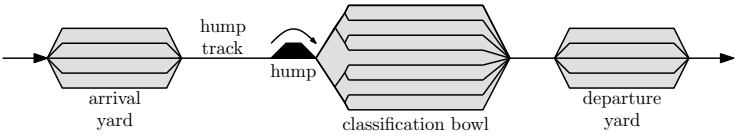


Figure 4.1: A schematic layout of a typical hump yard.

or *shunting* yards). The process of train classification is often the bottleneck of the freight transportation and naturally limits the throughput in the network. Hence, planning has to be made with respect to the capacities of the involved classification yards.

In this chapter, we study one specific process of forming outbound trains from the cars of inbound trains within a classification yard. In particular, we do not require a particular order of the cars of trains departing form the classification yard.

A classification yard typically consists of an *arrival yard*, a *hump track* with a *hump*, a *classification bowl*, and a *departure yard*. Therefore, classification yards with a hump are often called *hump yards*. A typical layout of a hump yard is illustrated in Figure 4.1. The *arrival yard* is a composition of tracks used for storing cars of inbound trains, where all cars of an inbound train are decoupled and stored on a single track. Similarly, a *departure yard* is a composition of tracks for storing outbound trains. The outbound trains are formed in the *classification bowl*, which consists of *classification tracks*. The hump is connected to the classification tracks by a set of switches such that a car or an engine can reach any classification track from the hump. To form the desired composition of cars on the classification tracks, two operations are used: the *pull-out* and the *roll-in* operation. In the pull-out operation, the cars of a specified classification track are coupled and pulled by the engine over the hump onto the hump track; there, the cars are decoupled and are ready for the subsequent roll-in. During the roll-in operation, the (decoupled) cars on the hump track are pushed over the hump and roll into the classification bowl by means of gravity only; each individual car is guided to a desired classification track by appropriately setting the switches of the classification bowl. In a hump yard, a pull-out operation is always followed by a roll-in operation. After the formation of an outbound train is finished, its cars are coupled and the train is moved by an engine to the departure yard.

Given the arrival and departure times of the inbound and outbound trains, respectively, as well as the cars belonging to these trains, the *operational plan* of a hump yard needs to decide the movements of every car in the hump yard by the means of roll-ins and pull-outs in order to achieve the desired formation of every outbound train before its departure time. There may be various constraints on the formation of the outbound trains. In particular, there are situations when the order of cars within a train is important. In this chapter we only consider the case in which no particular order of cars within an outbound train is required. This is usually the case for freight trains which are not delivering goods to their respective final destination but to another classification yard. A general operational plan would also need to decide the time points when cars enter (from the arrival yard) and leave (to the departure yard) the classification bowl. We sketch a heuristic on how to obtain such a schedule in Section 4.5.2. In the remainder of this chapter we assume that these time points are given. Hence, we focus solely on planning the movements of cars within the classification bowl. At present, operational plans are hand-made. In order to compute such plans automatically, we present several results in this chapter, including our modeling approach, relation to algorithmic theory, and initial computational results suggesting that operational plans can be found in less than 20 minutes.

The particular class of operational plans which we study in this chapter is a generalization of current customs that are used in several hump yards in Europe. In these operational plans, the majority of classification tracks is reserved for at most one outbound train at any point in time. A few remaining classification tracks, called *mixed tracks*, are used to store a mix of cars (which may arrive long before their planned departure) of different outbound trains in order to increase the capacity of the classification bowl. These mixed tracks are pulled-out, e.g., at some fixed time-points per day, in order to distribute cars onto classification tracks. A car can only go to its track once it is reserved for its outbound train. The remaining cars go back to the mixed tracks.

The chapter is structured as follows. We first define the mixing problem formally in Section 4.1.1, and then review the related work and the best practice in Section 4.1.2. We connect the mixing problem to various coloring problems of intervals in Section 4.2. We build upon these results in Section 4.3 where we also develop two heuris-

tics for the problem. We then present our mixed-integer program for the mixing problem in Section 4.4. The experimental results are discussed in Section 4.5.

4.1.1 Problem Definition

We consider a classification bowl consisting of k classification tracks $\kappa_1, \ldots, \kappa_k$, where a classification track κ has length ℓ_κ. Furthermore, some of the tracks of the classification bowl are used for mixing cars of different outbound trains. We denote the sum of their lengths as ℓ^{mix} and refer to them simply as *the mixed track*. There are n^{in} inbound trains, which are to be formed into n outbound trains, where each car of an inbound train belongs to exactly one outbound train. Individual cars having the same inbound and outbound trains are handled as a single unit, a car *group* g, which has physical length ℓ_g. Hence, an outbound train j consisting of a set $\mathcal{G}_j$ of car groups has length $\sum_{g \in \mathcal{G}_j} \ell_g$. For each outbound train j we need to *assign* a classification track κ to j on which it will be formed. The track κ has to have sufficient length, i.e., the length ℓ_κ must be at least the length of the outbound train. Every outbound train j has a fixed time o_j when it leaves the classification bowl (to the departure yard). Thus, by this time, all cars of the train j have to be on its assigned track. For each outbound train j we need to decide a time interval (s_j, o_j) during which the respective assigned track κ is reserved solely for cars of train j. Every inbound train i has a roll-in time r_i at which the cars of the inbound train are rolled in (from the arrival yard) over the hump (into the classification bowl). Each car group g is rolled-in either to the mixed track, or to the classification track κ which is assigned to the outbound train j to which group g belongs: if at time r_i the assigned track κ is already reserved for outbound train j (i.e., if $r_i > s_j$) then group g is rolled-in to the classification track κ, otherwise it is rolled-in to the mixed track. A car that is rolled-in to the mixed track needs to be pulled-back from the mixed track over the hump and rolled-in to the assigned track during the time interval that is reserved for its outbound train on that track. For this purpose, the mixed track is pulled out at fixed times $p_1, \ldots, p_m$. At such a time, all cars of the mixed track are subsequently rolled-in either to a classification track (if the respective assigned track has been already reserved), or back to the mixed track (if the respective track has not yet been reserved). We call each time interval between two consecu-

tive pull-outs a *period*.

In this chapter we consider the problem of assigning a classification track κ to every outbound train j, as well as deciding the time s_j when the assigned classification track should be reserved for train j, such that all outbound trains can be formed and leave the classification bowl on time. We will refer to this problem as the *mixing problem*. Observe that the schedule of the hump, which specifies the times of the roll-in and pull-out operations as well as the time o_j of each outbound train j, is fixed. Thus, the set of cars which are stored on the mixed track is determined by the choice of s_j for each outbound train j. A *feasible track allocation* is a solution of the mixing problem such that each train fits on its assigned track. Note that a feasible track allocation does not necessarily respect the capacity of the mixed track.

4.1.2 Related Work

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

Many research efforts related to the operation of classification yards have been put in *sorting schemes* for sorting cars inside a classification bowl. Given a sequence of n cars labeled from 1 to n, the general goal of a sorting scheme is to form, by roll-in and pull-out operations, a sorted sequence of cars. Early literature considers sorting schemes that essentially perform the same sorting steps for any input sequence of a given length [79]. More recently, it has been studied how to utilize the "pre-sortedness" of the input in order to minimize the number of pull-out operations [20, 53], as well as variants thereof [19]. A recent survey by Gatto et al. [44] gives an overview of this topic.

A related problem is the parking of trams in the evening on tracks in depots such that the trams can leave the depot in the morning without any shunting operation [9, 22, 86]. Another related problem is the train scheduling at yards, i.e., the problem of assigning trains and train times for a set of rail lines and station stops. This problem was considered by He et al. [50], together with some operational planning

at classification yards in China, although under different considerations than in our case.

In the problem considered in this chapter, we have to decide for each outbound train both the classification track on and the time at which it will be formed. A related problem for passenger trains has been considered [16], in which more than one inbound train can be assigned to a track of a train station. The problem asks for an assignment of tracks to inbound trains such that the trains do not block each other when departing the train station.

Best Practice Today, the planning of hump yard operations is to a large extent done manually. In the Hallsberg yard in Sweden, where the data for our experimental evaluation was collected, detailed planning is done by the hump-yard staff one day at a time, usually during the morning when fewer trains arrive than in the afternoon. The allocation of tracks at the arrival yard and departure yard is performed manually, independently from other operations, and in advance by traffic-planning personnel, who are not directly involved with hump yard operation and detailed planning. However, frequent communication between the different groups happens, since the allocation of arrival and departure yards and the yard operation planning is interdependent and cannot be done in full isolation. The typical practice at Hallsberg is to use the same roll-in order as the arrival time order and the same roll-out order (onto the departure yard) as the departure time order. In this chapter we follow this practice in that we assume the roll-in times and roll-out times (into and out of the classification bowl, respectively) to be given as part of the input. For the experimental evaluation where we do not have this data, we compute these times as described in Section 4.5.2. Also, a common practice is to pull-out all mixed tracks together, although it could be beneficial to pull-out the mixed tracks independently. For our purposes, we treat these mixed tracks as one virtual mixed track, where we set the length of the track and the duration of a pull-out operation accordingly.

4.1.3 Summary of Results

We study the problem of allocating classification tracks to outbound trains such that every outbound train can be built on a separate classification track. We observe that the core problem can be formulated

as a special list coloring problem in interval graphs, which is known to be NP-complete. We focus on an extension where individual cars of different trains can temporarily be stored on a special subset of the tracks. This problem induces several new variants of the list-coloring problem, in which the given intervals can be shortened by cutting off a prefix of the interval. We show that in case of uniform and sufficient track lengths, the corresponding coloring problem can be solved in polynomial time, if the goal is to minimize the total cost associated with cutting off prefixes of the intervals. Based on these results, we devise two heuristics as well as an integer program to tackle the problem. As a case study, we consider a real-world problem instance from the Hallsberg hump yard in Sweden. In our experiments, we plan over horizons of seven days. We obtain feasible solutions from the integer program in all scenarios, and from the heuristics in most scenarios.

4.2 Relation to Interval-Coloring Problems

The mixing problem can be seen as a family of specific coloring problems of intervals. In this section, we give complexity results based on these relations and devise two heuristics for our problem that we experimentally evaluate on real-world data in Section 4.5.

Recall that in the mixing problem we are asked to determine for each outbound train i a track κ and a time interval I'_i, during which the track is reserved exclusively for the formation of that train. Observing the roll-in times of the inbound trains, we can obtain for every outbound train i a time interval $I_i = (arr_i, dep_i)$ in which cars of the outbound train arrive to the classification bowl, i.e., $arr_i = \min_{g \in \mathcal{G}_i} r_g$ (where r_g is the roll-in time of the inbound train to which car group g belongs, and $\mathcal{G}_i$ is the set of car groups of train i), and $dep_i = o_i$ is the time when the train i leaves from its classification track to the departure track. Thus, without loss of generality, the time interval I'_i is a sub-interval of I_i of the form $I'_i = (arr'_i, dep_i)$, $arr'_i \geq arr_i$, i.e., we *cut off a prefix* of I_i to obtain I'_i. Note that all cars that have their roll-in scheduled before the start of I'_i have to be send to the mixed track. Further, note that we cannot cut off an arbitrary prefix: whenever $arr'_i \neq arr_i$, there has to be a pull-out of the mixed track between arr'_i and dep_i, because we require that every car that is sent to the mixed track is at some point brought to the actual track κ (before the train departs). This, together with some

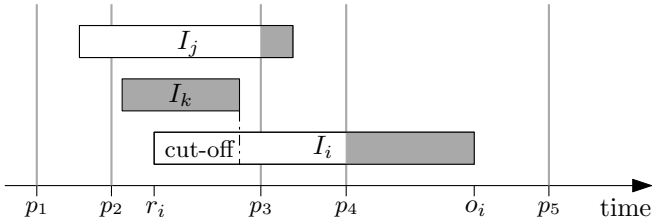


Figure 4.2: Three outbound trains i, j and k induce the (mutually intersecting) intervals I_i, I_j and I_k, depicted as rectangles. The pull-out of the mixed tracks happens at times $p_1, \ldots, p_5$. The uncuttable part of an interval (depicted in gray) is defined by the last pull-out time of the mixed track that is contained in the interval. If we assume two available tracks, then the three intervals cannot be assigned to the tracks without cutting off. Here, a cut-off of the interval I_i at the end point of I_k allows a placement of I_i' and I_k on the same track. For simplicity, technical setup times have been omitted in this figure.

technical setup times which we do not describe here for simplicity, induces for every interval I_i an *uncuttable part* of I_i, i.e., a suffix of I_i during which all not yet rolled-in cars of the outbound train have to be rolled in directly onto its classification track. The requirement that every track κ is at any time reserved for at most one train translates into the condition that, whenever trains i and j are assigned the same track κ, the corresponding intervals I_i', I_j' do not overlap. Here and in the following, two intervals overlap if they intersect in more than one point. Figure 4.2 illustrates our discussion.

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

In general, the list-coloring problem of intervals is NP-complete. In our case, the lists do not have arbitrary structure: Assume w.l.o.g.

that the classification tracks $\kappa_1, \ldots, \kappa_k$ are ordered increasingly by length. For each outbound train i, let $\mu(i)$ indicate the smallest track on which it fits (we assume that every train fits on the longest track κ_k); the list L_i is then just $\{\kappa_{\mu(i)}, \ldots, \kappa_k\}$. The resulting special list-coloring problem is called the *μ-coloring problem*, which is known to be NP-complete even for interval graphs, see [12]. As a consequence we immediately obtain the following theorem:

Theorem 4.1. *Finding a feasible track allocation for the mixing-problem is NP-complete even for instances where the capacity of the mixed track ℓ^{mix} is zero, or where ℓ^{mix} is unlimited and all intervals may have arbitrary uncuttable parts.*

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

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

If we assume that both the mixed track and all the classification tracks have sufficient length to store all cars and trains, respectively, a natural objective for the mixing problem is to minimize the total number of cars which are sent to the mixed track, subject to a feasible track allocation. In the following we show that this problem can be solved in polynomial time. Further below, we will use this result to devise an improvement heuristic for the mixing problem.

Theorem 4.2. *In case of uniform and sufficient track lengths, the problem of finding a feasible track allocation that minimizes the sum*

of all cars sent to the mixed track over all time periods is solvable in polynomial time.

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

4.3 Heuristics for the Mixing Problem

In the following we present two heuristics based on observations of the previous section. The first heuristic is a construction heuristic that aims at finding a feasible track allocation. The second is an improvement heuristic that, given a feasible track allocation, finds a local optimum with regard to the total number of extra roll-ins.

4.3.1 A Construction Heuristic

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

Coloring Intervals We first color the intervals in a greedy way, mimicking the greedy coloring of interval graphs without lists. We assume that we have infinitely many colors available (i.e., not only k). We start by sorting the intervals in a non-decreasing order of their starting time point. We color the intervals in this order and assign to each interval I_i the smallest *non-conflicting color* from its list L_i, i.e., the smallest color $c \geq \mu(i)$ such that no interval to which color c has so far been assigned overlaps with I_i.

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

Cutting off Intervals If there is an interval I_i that is colored with a color $c > k$, then I_i overlaps with intervals that are together assigned every color in $\mu(i), \mu(i)+1, \ldots, k, \ldots, c$. These intervals mutually intersect and thus form a clique K. We find a maximal

clique containing an interval of the largest assigned color c. Let $q := c - k$, i.e., the number of intervals of K that use a color $c > k$. The heuristic tries to reduce the size of the clique by cutting off q intervals of K, in order to be able to increase the set of available colors for each of those intervals.

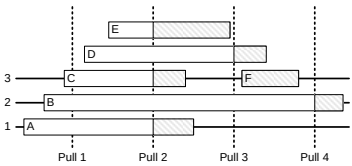
The cut-offs are computed as follows. First, the intersection of all intervals in K is computed, which is an interval by itself. Let t denote the end of this interval. If possible, cut off q of the intervals of K at point t. In particular, cut those q intervals of K that minimize the resulting additional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

This heuristic is illustrated in Figure 4.3. Clearly, the heuristic may fail to produce a feasible list-coloring. Further, note that even if a feasible list-coloring can be found, the heuristic does not guarantee to find a track allocation that is feasible with regard to the capacity of the mixed tracks.

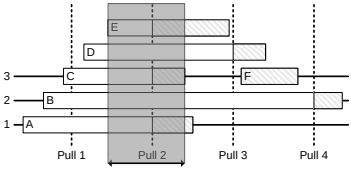
4.3.2 An Improvement Heuristic

Once a feasible assignment of tracks to outbound trains exists, one can furthermore try to improve the solution towards a local optimum. In particular, we are interested in minimizing the total number of cars (over all periods) sent to the mixed track, which we call the number of *extra roll-ins* for short, as will be motivated in Section 4.4.2.

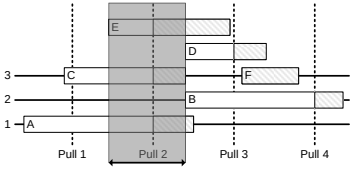
The heuristic is based on two observations. First, when looking at a feasible solution, one observes that both the tracks and the trains can be partitioned into subsets, called *buckets*, such that each train in a bucket fits on all tracks in the same bucket. Secondly, given a feasible solution, one can minimize the mixed-track usage for each bucket of tracks independently. Because within a bucket, all assigned trains fit on all tracks, it suffices to solve an assignment problem for each bucket, as detailed in Theorem 4.2, in order to find an optimal reassignment of trains to tracks within that bucket. Note that also this heuristic does not guarantee feasibility regarding the capacity of the mixed tracks.



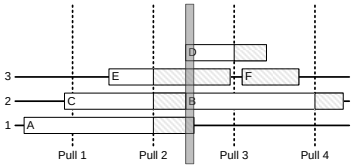
(a) Greedy coloring by start time.



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



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



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

Figure 4.3: First steps of the interval coloring heuristic involving trains A-F, and three classification tracks. Gray areas of the intervals indicate allocations which cannot be cut-off, as there is no pull-out (dotted vertical lines) of the mixed track. The dark grey rectange depicts the intersection of the respective clique of intervals.

4.4 Integer Programming Model

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

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

4.4.1 Capacity of the Mixed Tracks

To limit the amount of used mixing capacity, we first note that the set of mixed cars can only change when roll-in or pull-out operation is performed. In addition, a car group that is send to the mixed track at a roll-in at time s will stay mixed at least until the first pull-out that is scheduled after s, which we denote by p_s^+. Therefore, it suffices to ensure feasibility of the mixed track usage at the end of each period, when the maximum usage within that period is attained. Let

$$\mathcal{X}_{is} = \{g \mid g \in \mathcal{G}_i, r_g < s\}$$

be the set of groups of cars of an outbound train i that are mixed as a result of i starting at time s. We now need to consider which group of cars in train i are mixed at a certain pull-out p, given a start time s for i. To determine this we will check whether the prefix of I_i that is cut-off contains p, and which groups from $\mathcal{X}_{is}$ are rolled in before p.

Let $\mathcal{P}_i$ be the set of *relevant pull-outs* for train i, which occur after the first group roll-in and before the time the train needs to start preparations for departure. Furthermore, let

$$\mathcal{S}_i = \mathcal{P}_i \cup \{r_g \mid g \in \mathcal{G}_i \wedge r_g < \max \mathcal{P}_i\}$$

be the set of *relevant start-times* for train i, i.e., the union of all relevant pull-outs for train i and those roll-in times of groups of i which are scheduled before the last relevant pull-out for i. Now, if the allocation of the classification track for train i starts at time $s \in \mathcal{S}_i$, then

all groups $g \in \mathcal{X}_{is}$ have to be mixed and hence stay on the mixed track until $p_s^+ \in \mathcal{P}_i$, which is the first pull-out after s. Therefore, a group g will be mixed during the period ending at p if $r_g < s$ and $r_g < p \leq p_s^+$. Formally, we let

$$\mathcal{A}_p = \{(i, s) \mid 1 \leq i \leq n, p \in \mathcal{P}_i, s \in \mathcal{S}_i, p \leq p_s^+\}$$

be the set of pairs of all trains and start-times possibly affecting a pull-out p, where n is the number of outbound trains. Given $\mathcal{A}_p$ we can now define the mixed capacity constraints as shown below in Equation (4.5). Informally, this equation states that for all trains i and start times s affecting a pull-out p it holds that if the reservation of the classification track of i starts at s then the total length of the groups that arrive before p may not exceed the mixing capacity ℓ^{mix}.

4.4.2 Counting Extra Roll-ins

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

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

4.4.3 An Integer Programming Formulation

We are now ready to formulate the full integer programming model, including sequencing constraints on the classification tracks. We use binary variables x_{is} that indicate at which time s the reservation of a

classification track for outbound train i starts, as well as binary variables $y_{i\kappa}$ that indicate whether the outbound train i is allocated to track κ.

$$\min \quad \sum_{i=1}^{n} \sum_{s \in \mathcal{S}_i} c_{is} x_{is} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{\kappa \in L_i} y_{i\kappa} = 1, \qquad 1 \leq i \leq n \tag{4.2}$$

$$\sum_{s \in \mathcal{S}_i} x_{is} = 1, \qquad 1 \leq i \leq n \tag{4.3}$$

$$\sum_{\substack{s \in \mathcal{S}_j \\ s < o_i}} x_{js} + y_{i\kappa} + y_{j\kappa} \leq 2, \qquad \substack{(i,j) \in IJ \\ \kappa \in L_i \cap L_j} \tag{4.4}$$

$$\sum_{(i,s) \in \mathcal{A}_p} \sum_{\substack{g \in \mathcal{X}_{is} \\ r_g < p}} \ell_g x_{is} \leq \ell^{\text{mix}}, \qquad p \in \mathcal{P} \tag{4.5}$$

$$x_{is}, y_{i\kappa} \in \{0, 1\} \tag{4.6}$$

Equation (4.1) gives the objective in terms of the number c_{is} of extra roll-ins due to mixing, which results from using start time s for train i. Equation (4.2) ensures that all trains i are allocated to a track from the set L_i of feasible (w.r.t. length) tracks for i, and Equation (4.3) ensures that each train i has a start time s from its set of relevant start-times $\mathcal{S}_i$. Equation (4.4) states that for each pair of trains (i, j), where i leaves its classification track before j, but departs after the first group of cars of j is rolled-in, either i and j are on different tracks, or j starts its allocation of the common track κ after i has left. Formally, we define

$$IJ = \left\{ (i,j) \mid 1 \leq i < j \leq n \wedge \min_{g \in \mathcal{G}_j} r_g < o_i \right\} \quad ,$$

where we assume that the outbound trains are indexed according to ascending departure times from the classification bowl. Finally, by Equation (4.5) we ensure that the mixing capacity limit is respected in each period.

In the experiments below, we will refer to the above integer program (4.1) to (4.6) as the *optimization problem*. In order to facilitate finding a feasible solution, we further define the *feasibility problem*

as the integer program consisting of Equations (4.2) to (4.6) with an additional continuous non-negative variable v representing "virtual mixing capacity". The variable v is then added to the right-hand side of Equation (4.5). By replacing the objective Equation (4.1) with

$$\text{minimize } v,$$

one seeks to obtain a feasible solution to the mixing problem without specifically minimizing the number of extra roll-ins.

Let us conclude this section with some remarks about the integer programming formulation above. Note that this formulation is in fact a binary program. In an earlier version of this work [10], we used a mixed integer programming formulation instead. In our experiments, both formulations turned out to have very weak lower bounds. The advantage of the binary formulation, though, is that the standard solver we used could quickly find feasible solutions (by its generic heuristics), whereas finding feasible solutions in the mixed integer formulation (with many "big M" constraints) turned out to be slow and less reliable. Besides, the above formulation is much shorter and captures more of the structural insights into the problem.

4.5 Case Study

The Swedish traffic administration authority has provided us with historic data for the Hallsberg Rangerbangård hump yard in central Sweden for validation of our approach. Hallsberg has 8 tracks of length 595 to 693 meters on the arrival yard, two parallel humps (of which only one is in use), 32 available classification tracks of varying length (between 374 and 760 meters), and 12 tracks with length 562 to 886 meters on the departure yard. Although there are several other tracks on the yard (most notably tracks going to light and heavy repair facilities) they are not normally used for shunting, and we are therefore not considering them in the model. The layout of Hallsberg is shown in Figure 4.4.

We used the following working process to determine a shunting plan: First, preprocess the data according to Section 4.5.1. Second, determine suitable roll-in (r_i), pull-out (p_j) and roll-out times (o_j) for all inbound trains i and outbound trains j by the method described in Section 4.5.2. Third, separate the problem data into instances containing all rolled-in trains in a single week, ranging from Saturday

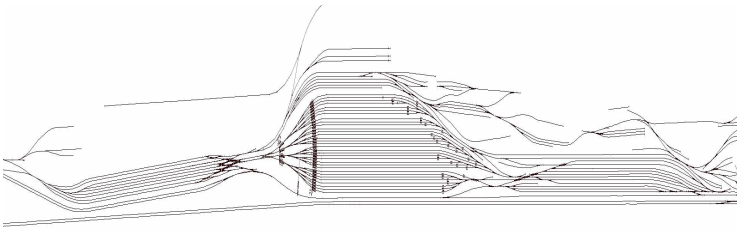


Figure 4.4: Layout of the Hallsberg classification yard in Sweden. The arrival yard is on the left, followed by the hump, the switching system, classification tracks, and finally the departure yard on the right. Image from Handbok BRÖH 313.00001, Anläggningsbeskrivning Hallsbergs rangerbangård. The image is scaled to emphasize details.

until the next Friday, using the roll-in and pull-out times obtained in the second step. Fourth, solve the mixing problem instances using the construction heuristic from Section 4.3.1 and the improvement heuristic from Section 4.3.2. Fifth, also solve the feasibility mixing problem instances using the MIP model from Section 4.4.3, minimizing the virtual mixing capacity v. Finally, improve the mixing solutions obtained from the MIP model by fixing v at zero and solving the original integer program (4.1) to (4.5).

4.5.1 Preprocessing Traffic Data

We collected five months of historic traffic data for the yard, including all inbound trains, outbound trains, and the set of cars going from each inbound train to each outbound train. The data was taken from the period from December 11, 2010 to May 10, 2011, which contains intervals of both high and low activity, such as a longer holiday period and at least one major traffic disruption. Timing parameters, such as setup times, durations of roll-ins, etc., were chosen according to [6]. Cars with a local source or destination were not included in the data set and were therefore not considered.

The data we collected contain for every inbound train only the time when it arrives at the arrival yard. The time when the cars of the

inbound train roll-in to the classification bowl are not available. Further, the data did not contain the time points when the mixed tracks were pulled-out. Therefore, we had to compute a hump schedule for all roll-in and pull-out operations to complete the input for the mixing problem of the previous section. This will be described in Section 4.5.2.

The data contain further ambiguity regarding the matching of cars from inbound to outbound trains. In principle, a car can be matched to any outbound train that stops at the destination of the car. In the data, only one such outbound train is listed for each car. In some cases, this train did not comply with the minimum time required to process a car. There is also ambiguity regarding the actual arrival and departure times for some of the trains. Finally, data for trains arriving or departing outside the period for which the data was provided are missing.

In order to resolve this ambiguity, we proceeded as follows. For our experiments, we required a minimum time span of 180 minutes between arrival at and departure from the shunting yard (i.e., not only the classification bowl). Further, we required that cars spend at most 48 hours on the yard. Finally, we required that trains do not exceed the length of the longest available classification track. Car records that could not be matched this way were discarded. In total, 3594 arrivals, 3654 departures and 17684 car groups were handled. Inbound trains vary in length between 12.8 and 929 meters, outbound trains between 12 and 1252 meters. For five outbound trains we had to discard some of their groups in order to stay below the maximal track length of 760 meters.

4.5.2 Computing the Missing Hump Schedule

The mixing problem is given by the roll-in times of the inbound trains, by the departure times of the outbound trains from the classification bowl, and by the times of the pull-outs. The traffic data provided to us does not contain this information: only the arrival times to the arrival yard and the departure times from the departure yards are known. For the experiments, we therefore had to compute the missing data, trying to mimic the current practice in the yard, as described in the following. Because the pull-backs as well as the roll-ins occupy the hump, we need to schedule the roll-ins and pull-backs such that no two such

actions on the hump overlap in time. Additionally, we want to find a set of pull-outs such that for each outbound train there is the possibility to store some of its cars on the mixed track. We achieve this by creating a rough assignment of trains to classification tracks in a round-robin fashion (and ignoring mixing capacities), and computing a minimum set of pull-outs, such that there is one suitable pull-out for each train that needs to store cars on the mixed tracks in this round-robin schedule.

We assume that all tracks on the arrival and departure yards can accommodate all inbound and outbound trains. We determine the roll-in times and pull-out times sequentially by first considering the roll-ins, and then by inserting pull-outs at suitable time points. We omit a detailed description of the preprocessing. We remark that using this preprocessing we had to delay a small fraction of the trains in the five month period (0.8 % of the inbound trains for in total 82 minutes and 0.08 % of the outbound trains for in total 70 minutes). It should be noted, however, that arrival and departure times can be negotiated with the network provider. Further, in daily operations, the planners do not strictly follow a fixed rule to determine the roll-in order. In about 90% of the cases, trains are rolled-in in the same order as they arrive. In the remaining cases, the above rule of rolling in according to earliest deadline is followed.

4.5.3 Results

Table 4.1 shows the results obtained for the problem instances using the above approaches. The instances are seven days long and cover Saturday until the next Friday. In the table, *Heuristic* is the heuristic from Section 4.3.1, *Heuristic++* is the same heuristic improved using the algorithm in Section 4.3.2, and *MIP* is the integer programming model from Section 4.4, where we first solve the feasibility mixing problem as described above. The MIP computations were carried out using Python 2.6 and Gurobi 4.5 on a standard dual-core desktop computer. A time limit of 10 minutes for the feasibility problem and an additional 10 minutes for the optimality problem was imposed.

As can be seen in Table 4.1, both heuristics reach feasibility with regard to the total mixing capacity of 1217 m in 19 out of 21 problem instances. The improvement heuristic from Section 4.3.2 lowers the mixed track usage and number of extra roll-ins compared to the origi-

Table 4.1: Experimental results in maximum mixed track usage (MTU) with a limit of 1217 m, and the extra car-roll-ins needed due to mixing (ER). $\bar{x}$ is the arithmetic mean. Infeasible solutions (weeks 16 and 21) are shown in italics.

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

nal coloring heuristic from Section 4.3.1 in many instances. The MIP model finds feasible mixing solutions in all considered instances, in many cases with much fewer extra roll-ins compared to the heuristics. In addition, in 17 out of the 21 cases we could have managed with just one mixed track of length of 608 or 609 m instead of two. However, it should be noted that the runtime when optimizing the number of extra roll-ins was terminated after 10 minutes without finding an optimal solution. The reported MIP gap for the model minimizing the number of car roll-ins is almost always 100.0 %, i.e., there is no lower bound. At the same time, the run-time of the heuristics is negligible, as it is in the order of a fraction of a second. Although the experiments suggest that the MIP performs fast enough, we remark that faster algorithms would enable other applications as, e.g., online booking systems, similar to those in passenger traffic, where customers could make a reservation to book their cars on specific trains.

Worth noting is that instances 2–4 contain both holidays and major disruptions due to snowfall, which explains the lower load seen in Table 4.1. In addition, instance 7 contains a large derailment which occurred on a major line in northern Sweden. As a result, freight had to be transported on lines with lower capacity than normal, hence the lower volumes at Hallsberg during this period.

4.6 Conclusion

In this chapter, we studied the track allocation problem for hump yards where temporary storage (mixing) of cars is allowed. Given departure times of the outbound trains, a hump schedule specifying the times when cars enter the classification bowl, and times when the mixed track is pulled back, we assign and reserve to each outbound train a classification track so that the cars sent to the mixed track do not exceed the capacity of the mixed track.

We have demonstrated a connection of the mixing problem to various interval graph coloring problems, and presented a new class of interval-coloring problems where intervals can be cut-off, while obeying various constraints on the cut off parts. We have seen that two extreme cases, where the length of the mixed track is either zero or infinity, result in an NP-complete problem. We have also shown that for the goal of minimizing the sum of cars (car-meters) sent to the mixed track, the problem can be solved in polynomial time, assuming uniform lengths of the classification tracks. Based on this, we have developed and implemented two heuristics for the mixing problem. Further, we solve the mixing problem using a new integer programming model. The model produced week-long feasible plans for all problem instances tried in less than 20 minutes, which is quick enough for practical use. The lower bounds obtained by the linear programming relaxation seems to be weak, and optimality was only proven in few instances where freight volumes were lower than normal. Both the heuristics and the integer program have been experimentally tested on week-long instances taken from five months of real-world data from the Hallsberg yard in Sweden. For the experimental evaluation, we have also implemented the current best-practice for scheduling the hump.

A possible future research direction is to integrate the computation of the hump schedule (i.e., scheduling roll-ins and pull-outs) and the mixing problem. A further natural question is to incorporate the non-deterministic nature of delays of incoming trains into the model, in order to find plans that are in some sense robust against such delays.

Chapter 5

Sorting Cars at Classification Yards

In this chapter, we answer a question posed by Jacob et al. in [53] regarding the complexity of sorting cars in a hump yard when the order of incoming trains can be chosen. The authors prove a one to one correspondence of this problem to a special case of the minimum feedback arc set problem, namely restricted to directed multigraphs whose edges form a Eulerian path. Here, we prove that this problem is NP-complete.

Definition 5.1 (minimum feedback arc set (MFAS))**.** *Given a directed graph $G = (V, A)$ and a number $k \leq |A|$, is there a linear ordering $<_L$ of the vertices, such that the set $B := \{(v, u) \in A \mid u <_L v\}$ has cardinality of at most k?*

It is well known that MFAS is NP-complete [41, GT8]. In the following, we refer to a linear ordering $<_L$ as a *layout*, to the arcs of B as *backward arcs*, and to the remaining arcs $A \setminus B$ as *foward arcs*. We define a u-v-cut as the set of arcs $\delta(u, v) := \{(i, j) \in A \mid (i \leq_L u <_L v \leq_L j) \vee (j \leq_L u <_L v \leq_L i)\}$.

Theorem 5.2. *The minimum feedback arc set problem in directed multigraphs whose edges form a Eulerian path is NP-complete.*

Proof. By reduction from MFAS. Given a graph $G = (V, A)$ of an instance of MFAS, create a graph $G' = (V \cup \{v_0\}, A')$ that is a

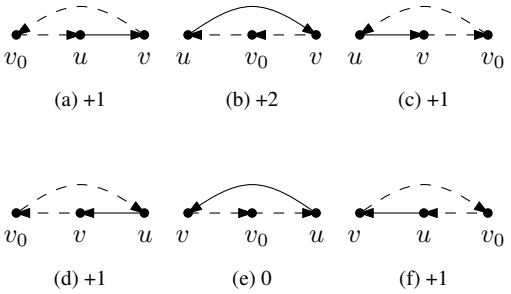


Figure 5.1: Reduction from MFAS. For each original (solid) arc (u, v) two additional (dashed) arcs are created. (a) to (c) show all possible layouts if (u, v) is a forward arc. (d) to (f) show all possible layouts if (u, v) is a backward arc. In each case, the number of additional backward arcs is indicated.

Eulerian path as follows. Let $A' = A$ initially. For each original arc $(u, v) \in A$, add the arcs (v_0, u) and (v, v_0) to A'. We prove that G has a minimum feedback arc set of size k if and only if G' has a minimum feedback arc set of size $k + |A|$.

Given an optimal layout of G with k backward arcs, a layout of G' with $k + |A|$ backward arcs can be obtained by placing v_0 before all vertices, i.e., by setting $v_0 <_L v$ for all $v \in V$, see Figure 5.1. Similarly, a feedback arc set of the same cardinality is obtained by placing v_0 after all vertices. It remains to show that placing v_0 between any pair of original vertices cannot result in a feedback arc set of smaller cardinality.

To that end, note that placing v_0 between the vertices of a forward arc $(u, v) \in A$ turns both additional arcs (v_0, u) and (v, v_0) into backward arcs, see Figure5.1(b). Placing v_0 between the vertices of a backward arc $(v, u) \in A$, however, turns both arcs (v_0, u) and (v, v_0) into forward arcs, see Figure5.1(e).

Compared to the case where v_0 is placed before all vertices, placing v_0 between two nodes u and v increases $|B|$ by one for each forward arc $(i, j) \in \delta(u, v)$ and decreases $|B|$ by one for each backward arc $(j, i) \in \delta(u, v)$

Observe that for a minimum feedback arc set it must hold that

there cannot be more backward than forward edges in any u-v-cut. Hence, placing v_0 between any two nodes u and v cannot result in fewer backward arcs as compared to placing v_0 in front.

Now, given an optimal layout of G' with $k + |A|$ backward arcs, it follows from the arguments above that v_0 must either be placed before or after all original vertices. In both cases, removing v_0 and all its incident arcs results in a layout of G with exactly k backward arcs. □

Part III

Theoretical Models for Dispatching

Chapter 6

Vertex Disjoint Paths in Planar Graphs

6.1 Introduction

Given a graph $G = (V, E)$ and a collection of pairs of vertices $\{s_i, t_i\}$, $i = 1, \ldots, k$, the vertex disjoint paths problem is to find k vertex disjoint paths $P_1, \ldots, P_k$, such that P_i is an s_i-t_i path, for $i = 1, \ldots, k$. In this chapter, we study variants of the vertex disjoint paths problem in planar graphs where, in addition to the aforementioned conditions, each s_i-t_i path must be selected from a given set of alternative paths.

Our original motivation to study these problems arises from routing trains through railway stations, see the preliminary version of this chapter [29]. A railway station may be modeled as a graph with nodes representing points on the tracks and edges representing track segments that connect such points. Two trains are in conflict if their routes share a point on the tracks. Hence, conflict-free routes correspond to vertex disjoint paths. Not every route which is physically feasible is desirable in practice, though. Therefore, railway planners usually allow only a small set of alternative paths for each train. This leads to various vertex disjoint paths problems where for each terminal pair, corresponding to a train with a given start and target location in the network of the railway station, a path has to be chosen from a given set of paths, i.e., from the possible set of routes for the

train. The abstractions made in this chapter neglect several important issues of the railway system (in particular regarding aspects of time) and thus our results are more of theoretical than practical interest. Nevertheless, it would be interesting to apply our results in the approximative evaluation of capacity in infrastructure planning. Finally, we think that the setting of predefined paths is a natural variant of disjoint paths problems and of general interest.

6.1.1 Problem Definition

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

An input instance for the problems we study is a triple $(G, T, \mathcal{P})$, defined as follows: $G = (V, E)$ is an undirected plane graph, i.e., a planar embedding of a planar graph G. T is a collection of k pairs of vertices $\{s_i, t_i\} \subseteq V$, $i = 1, \ldots, k$. $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$ is a collection of sets of paths, where $\mathcal{P}_i$, $i = 1, \ldots, k$, is a set of paths from s_i to t_i. We sometimes refer to the paths of a set $\mathcal{P}_i$ as *alternative* paths. The pairs in T are called *terminal pairs*, and their vertices are called *terminals*. A vertex may be a terminal of several terminal pairs. Two paths P_i, P_j are said to be *vertex disjoint* if $P_i \cap P_j = \emptyset$, so in particular they may not share a common terminal. A path from s_i to t_i is called an s_i-t_i path. The planar embedding of G separates the plane into distinct regions, called *faces*, bordered by the edges of the graph. The unbounded face of the graph's embedding is called the *outer face*.

We denote by p the maximum cardinality of a set in $\mathcal{P}$, so $p := \max_{1 \leq i \leq k} |\mathcal{P}_i|$. We denote by $\mathcal{R}$ the union of all sets of the collection, thus $\mathcal{R} := \bigcup_{i=1\ldots k} \mathcal{P}_i$ is the set of all given paths. Given a set of paths $S \subseteq \mathcal{R}$ we say that a function $c : S \to \{1, \ldots, r\}$, $r \in \mathbb{N}$ is a *proper coloring* of S if any two paths $P, P' \in S$ that intersect are assigned a different value by c, i.e., if $P \cap P' \neq \emptyset$ then $c(P) \neq c(P')$. We say $c(P)$ is a *color* of P. We study the following algorithmic problems:

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

Maximization Problem: Find a set $S \subseteq \mathcal{R}$ of maximum cardinality

such that the paths of S are vertex disjoint (thus $|S \cap \mathcal{P}_i| \leq 1$ for all $i = 1, \ldots, k$).

Routing-in-Rounds Problem: Find a set $S \subseteq \mathcal{R}$ with $|S \cap \mathcal{P}_i| = 1$ for all $i = 1, \ldots, k$, and a proper coloring $c : S \to \{1, \ldots, r\}$ minimizing r.

Clearly, the decision problem can be seen both as the maximization problem, where we are to decide whether there is a vertex disjoint set $S \subseteq \mathcal{R}$ of cardinality k, and as the routing-in-rounds problem, where we are to decide whether one color (i.e., round) suffices.

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

First, we consider the case where the terminals appear on the boundary, in a counterclockwise traversal of the boundary, as a sequence $s_1, s_2, \ldots, s_k, t_{\pi(1)}, t_{\pi(2)}, \ldots, t_{\pi(k)}$ for some permutation π. We say that such an instance has a *separating cut*, or that the *terminals can be separated*. See Figure 6.2 for an example.

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

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

This chapter is structured as follows: We discuss variants of the

decision problem in Section 6.2, of the maximization problem in Section 6.3, and of the routing-in-rounds problem in Section 6.4. An overview of the most important complexity results of this chapter is given in Table 6.1.

6.1.2 Related Work

Finding vertex disjoint s_i-t_i paths for given terminals $\{s_i, t_i\}$, $i = 1, \ldots, k$, in a given graph G is a well studied problem. Observe the crucial difference to our setting: there the chosen s_i-t_i path can be an arbitrary s_i-t_i path (not limited to be from $\mathcal{P}_i$). The vertex disjoint paths problem is NP-complete if k is part of the input, even when restricted to planar graphs, both in the directed and undirected case [65]. On the other hand, if k is not part of the input, then the problem is solvable in polynomial time in undirected graphs [72], in directed planar graphs [76], and in directed acyclic graphs [34]. In general directed graphs, however, the problem is NP-complete already for $k = 2$ [34]. Several variants of the problem have been studied. For example, the problem where the chosen s_i-t_i paths are asked to be shortest s_i-t_i paths has been studied in [56]. Practically efficient algorithms for special cases of the problem which are motivated by VLSI-layouts are surveyed in [71].

Observe that for directed graphs the vertex disjoint paths problem problem can be reduced to the edge disjoint paths problem. The edge disjoint paths problem has been studied in the classical scenario, i.e., where paths are not restricted to be chosen from sets of alternative paths, with respect to decision, maximization and routing in rounds setting. A survey on the decision problem can be found in [39]. The maximization problem has been studied in terms of the well known multicommodity flow problem, see, e.g., [3]. In [2], the routing in rounds setting is considered in terminology of communication networks where rounds are referred to as wavelengths. Recently, approximating the edge disjoint paths problem with congestion, i.e., where at most $c \geq 1$ paths may use an edge, has been studied, see, e.g., [7, 54].

Finally, we remark that the maximization problems that we study (i.e., the problems of the type M-VDP-*) can be seen as the problem of finding a maximum independent set in the conflict graph induced by the paths $\mathcal{R}$. In particular, the instances where the terminals lie

on the outer face (the problem M-VDP-OUT) form a class of conflict graphs that is equal to the class of *outerstring graphs*, for which the problem of finding a maximum independent set is an open problem. An *outerstring graph* is an intersection graph of *curves* lying in a disk where each curve has one endpoint on the boundary of the disk (in this setting, curves are also called *strings*). This class of graphs has been studied mainly from the graph-theoretic perspective [36, 37, 57, 58]. A significant difference in studying outerstring graphs and vertex-disjoint paths is that in M-VDP-OUT we are given the paths that form the intersection graph, whereas when studying outerstring graphs, such a representation is not necessarily available.

6.1.3 Summary of Results

We study variants of the vertex disjoint paths problem in planar graphs where paths have to be selected from given sets of paths. We investigate the problem as a decision, maximization, and routing-in-rounds problem. All considered variants are NP-hard in planar graphs. We prove, however, that instances with a separating cut are polynomially solvable for the decision and maximization versions of the problem. For the routing-in-rounds problem, we provide a p-approximation algorithm, where p is the maximum number of alternative paths for a terminal pair.

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

Table 6.1: Summary of complexity results, where k is part of the input.

6.2 D-VDP: Decision Problems

In this section we consider the decision version of the problem. An input instance is a triple $(G, T, \mathcal{P})$, where G is a plane graph, T is a collection of k terminal pairs $\{s_i, t_i\}$, $i = 1, \ldots, k$, and $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$, where $\mathcal{P}_i$ is a set of s_i-t_i paths. The problem is to decide whether there are k vertex disjoint s_i-t_i paths $P_i \in \mathcal{P}_i$, $i = 1, \ldots, k$.

We show that for planar graphs the general problem D-VDP-ANY is NP-complete whenever $p \geq 3$, where p is the maximal number of alternative paths per terminal pair, and solvable in polynomial time otherwise. The special case D-VDP-SEP, where the terminals can be separated, can be solved in polynomial time by reduction to M-VDP-SEP, for which a polynomial time algorithm exists as shown in Section 6.3.3.

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

Remark 6.1. *If there exists a solution for an instance of* D-VDP-OUT*, then the terminals must be nested.*

Next, we prove NP-completeness of D-VDP-ANY for $p \geq 3$ by reduction from PLANAR3SAT, which is defined as follows. Let $\varphi = (X, C)$ be an instance of 3SAT, with variable set $X = \{x_1, \ldots x_n\}$ and clauses $C = \{C_1 \ldots C_m\}$ such that each clause consists of exactly 3 literals. Define a formula graph $G_\varphi = (V, E)$ with vertex set $V = X \cup C$, and edges $E = \{(x_k, C_i) : x_k \in C_i \text{ or } \overline{x_k} \in C_i\}$. PLANAR3SAT is 3SAT restricted to instances φ for which G_φ is planar, and was proved NP-complete in [63].

Theorem 6.2. D-VDP-ANY *is NP-complete for* $p \geq 3$.

Proof. Let φ be an instance of PLANAR3SAT. To construct an instance of a graph $G_p = (V_p, E_p)$ for D-VDP-ANY, we start with $G_\varphi = (V, E)$. We substitute each node $C_i \in V$ by a corresponding

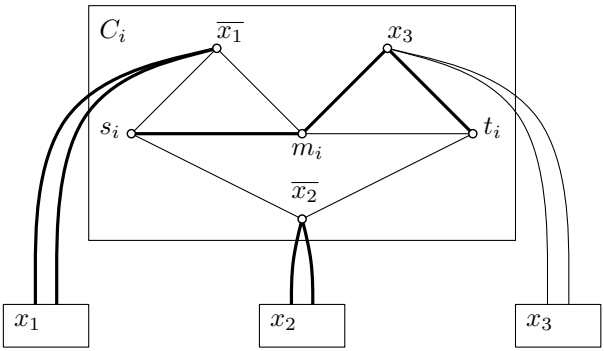
clause gadget, and each node x_i by a corresponding variable gadget, as described in the following.

A clause gadget as shown in Figure 6.1a is created for each clause $C_i \in \varphi$. It consists of 6 nodes. Let $C_i = \{l_1^i, l_2^i, l_3^i\}$, where l_j^i are the literals of C_i. Three nodes of the gadget correspond to these literals. They are connected to a path (s_i, m_i, t_i) in a way that depends on a plane drawing of G_φ. Let e_1, e_2, e_3 be the edges in a counterclockwise order connecting vertex $C_i \in V$ in G_φ with vertices $x_1, x_2, x_3 \in V$, where $l_1^i \in \{x_1, \overline{x_1}\}, l_2^i \in \{x_2, \overline{x_2}\}, l_3^i \in \{x_3, \overline{x_3}\}$. We add (l_1^i, s_i), (l_1^i, m_i), (l_2^i, s_i), (l_2^i, t_i), (l_3^i, m_i), (l_3^i, t_i) to E_p, as shown in Figure 6.1a. This gadget is planar. Moreover, if we substitute node $C_i \in G_\varphi$ with its clause gadget, literal nodes of the gadget are connected with corresponding variable nodes preserving the planarity of G_φ. We set $\{s_i, t_i\}$ as a terminal pair in G_p. We let $\mathcal{P}_i$ be the following set of alternative paths: $\{(s_i, l_1^i, m_i, t_i), (s_i, l_2^i, t_i), (s_i, m_i, l_3^i, t_i)\}$.

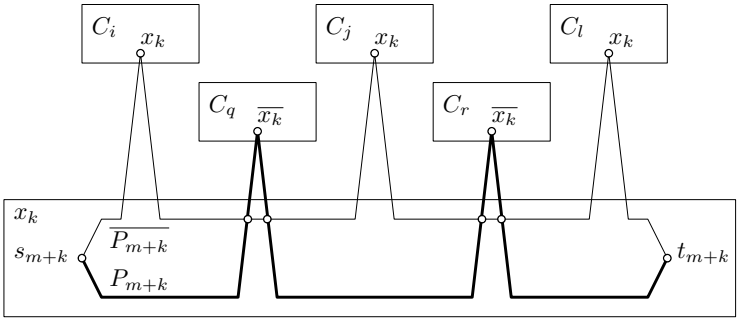
Now we construct a gadget for each vertex $x_k \in G_\varphi$. It consists of two terminal vertices $\{s_{m+k}, t_{m+k}\}$ and two alternative paths between them: $P_{m+k}, \overline{P_{m+k}} \in \mathcal{P}_{m+k}$. Path P_{m+k} contains all the literals $\overline{x_k}$ in the clause gadgets. We want to enforce that if the solution contains path P_{m+k}, then no other path (in a clause gadget) containing literal $\overline{x_k}$ can be chosen. Intuitively, choosing P_{m+k} corresponds to setting x_k to true, and choosing a path with x_k on it for a terminal pair of a clause gadget corresponds to satisfying the clause with literal x_k. Similarly, path $\overline{P_{m+k}}$ contains all the literals x_k in the clause gadgets. In order to draw path P_{m+k}, we substitute the edges that connect x_k with clause gadgets containing $\overline{x_k}$ by pairs of edges on the path from s_{m+k} to t_{m+k}. Thus, each such pair reaches the corresponding clause gadget. We proceed analogically to draw $\overline{P_{m+k}}$. Obviously, P_{m+k} can intersect $\overline{P_{m+k}}$, but in that case we add a vertex at the place of intersection to make G_p planar. The variable gadget is shown in Figure 6.1b.

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

Assume that $m + n$ disjoint paths, one for each terminal pair, can be chosen. To obtain a satisfying assignment for φ, set x_k to true if and only if P_{m+k} was chosen for terminal pair $\{s_{m+k}, t_{m+k}\}$. To see that each clause C_i is satisfied by that assignment, let $P \in \mathcal{P}_i$ be the



(a) Clause gadget with terminal pair. Example where x_3 satisfies the clause (due to choice of the bold s_i-t_i path).



(b) Variable gadget with terminal pair. Example where x_k is set to true (bold path).

Figure 6.1: Transformation from PLANAR3SAT to D-VDP-ANY.

path chosen for a terminal pair of the corresponding clause gadget, and let l_j^i be a literal of C_i lying on P. Assume w.l.o.g. that l_j^i is a non-negated variable x_j. In that case $\overline{P_{m+j}}$ could not have been chosen, and therefore x_j must have been set to true. Thus, clause C_i is satisfied by x_j.

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

In the following, we prove that we can solve instances having at most two paths per terminal pair in polynomial time by reduction to

2SAT, which is solvable in polynomial time, see e.g. [41].

Lemma 6.3. D-VDP-ANY *can be solved in polynomial time if* $p \leq 2$.

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

Observe that in the proof of the previous lemma we did not use that the underlying graph is a planar graph, and thus the problem where $p = 2$ can be solved for general graphs as well.

Corollary 6.4. *It can be decided in polynomial time whether there are vertex disjoint* s_i-t_i *paths* $P_i \in \mathcal{P}_i$, $i = 1, \ldots, k$, *where* $\mathcal{P}_i$ *is a set of at most two* s_i-t_i *paths in* any *graph* G.

6.3 M-VDP: Maximization Problems

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

Note that the problem can be seen as the problem of finding a maximum independent set in the intersection graph of the paths $\mathcal{R}$ (i.e., in a graph where nodes correspond to the paths and there is an edge between two nodes if the two corresponding paths intersect). To see this, observe that in any such independent set at most one s_i-t_i path can be chosen for every $i = 1, \ldots, k$. As we allow the terminals

to be at the same node, we can reduce the instances with $p > 1$ to (polynomially equivalent) instances with $p = 1$:

Remark 6.5. M-VDP-ANY *with k terminal pairs and at most p paths per terminal pair can be solved in polynomial time if and only if* M-VDP-ANY *with $k \cdot p$ terminal pairs and exactly 1 path per terminal pair can be solved in polynomial time.*

In the following, we first show that M-VDP-ANY is NP-complete. We leave the complexity of M-VDP-OUT open but show polynomial time solvability for the special case where paths in $\mathcal{R}$ have a certain monotonicity property. Finally, we consider M-VDP-SEP and show that it can be solved in polynomial time.

6.3.1 M-VDP-ANY: Terminals Anywhere

We show that M-VDP-ANY is NP-hard already for the case $p = 1$ (i.e., when there is one alternative path per terminal pair) by a reduction from D-VDP-ANY (with arbitrary p).

Theorem 6.6. M-VDP-ANY *is NP-hard already for $p = 1$.*

Proof. By reduction from D-VDP-ANY. Let $(G, T, \mathcal{P})$ be an instance of D-VDP-ANY, and denote by $(G, T', \mathcal{P}')$ the corresponding instance of M-VDP-ANY. For each terminal pair $\{s_i, t_i\} \in T$ we transform each path $P_i^j \in \mathcal{P}_i$, $j = 1, \ldots, |\mathcal{P}_i|$ into a terminal pair $\{s_i^j, t_i^j\}$ of T', where $s_i^j = s_i$ and $t_i^j = t_i$, and a set $\mathcal{P}_i^j$ of $\mathcal{P}'$ that consists of P_i^j only.

Clearly, every set of vertex disjoint paths in $(G, T, \mathcal{P})$ corresponds to a set of vertex disjoint paths in $(G, T', \mathcal{P}')$ of equal cardinality, and vice versa. Hence, there are k vertex disjoint paths connecting each terminal pair of $(G, T, \mathcal{P})$ if and only if the cardinality of an optimal solution for $(G, T', \mathcal{P}')$ is k. □

We remark that M-VDP-ANY remains NP-hard even for the case in which no two terminal pairs intersect in their terminals (by an easy modification of the proof above). It remains an interesting open problem to find out how good M-VDP-ANY can be approximated. We also note that contrary to the maximization problem, D-VDP-ANY with $p = 1$ is trivial to solve.

6.3.2 M-VDP-OUT: Terminals on the Outer Face

We leave the complexity of M-VDP-OUT open, and point to a similar open problem in graph theory, namely to the problem of finding a maximum independent set in outerstring graphs, e.g., see [36, 37, 58, 58]. An outerstring graph is the intersection graph of curves in a disk, where each curve has one endpoint on the boundary of the disk.

It is easy to see that the class of outerstring graphs is the same as the class of intersection graphs of the paths of instances of M-VDP-OUT with $p = 1$. Given a string representation of an outerstring graph, we can modify the representation of the graph such that every string is attached with both endpoints to the boundary of the disk: given any string with an endpoint in the interior of the disk, prolong the string by a curve that goes from that endpoint very closely along the string and ends close to the endpoint of the string on the disk's boundary. If we choose the curve to be close enough the new prolonged string does not intersect any new string. We can now construct a corresponding plane graph as an instance of M-VDP-OUT with $p = 1$: the disk of the outerstring graph representation forms the boundary of the outer face of our planar graph, the endpoints of each string correspond to a terminal pair in the outer face, and the string corresponds to the embedding of the path between the terminals; to make the graph planar, every intersection of the strings forms a vertex. Observe that there can be many intersections of the strings and thus, in general, this reduction does not guarantee that the size of the constructed graph for M-VDP-OUT will be polynomial.

Conversely, given a planar graph with terminals on the outer face, we can consider the boundary of the outer face as a disk and the s_i-t_i paths as strings that have both end points on the boundary of the disk.

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

Remark 6.7. M-VDP-OUT *with monotone paths (and* $p = 1$*) can be solved in polynomial time.*

Proof. By reduction to the maximum independent set problem in circle graphs, for which a polynomial time algorithm is given in [45]. A circle graph is the intersection graph of a family of chords in a circle.

Considering that the paths of the instance of M-VDP-OUT are monotone and have their ends on the outer face of the graph, it is easy to see that there is a family of chords of a circle where two chords cross iff their corresponding paths cross. Further, because $p = 1$, a maximum independent set in the corresponding circle graph corresponds to an optimal solution of the considered instance of M-VDP-OUT. □

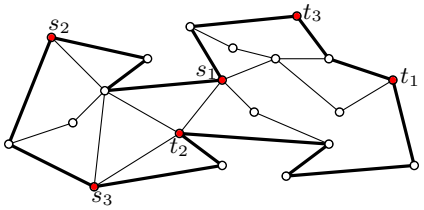
We note that this result does not necessarily hold for *monotone* outerstring graphs (where two strings intersect at most once), because a string does not have to be attached by both endpoints to the boundary of the cycle, and thus with monotone paths in planar graphs (where both end points are required to lie on the boundary of the outer face) we cannot mimic monotone strings.

6.3.3 M-VDP-SEP: Separating Cut

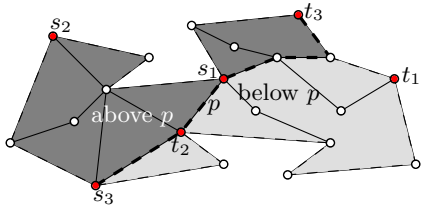
In this section we consider instances with a separating cut, i.e., instances where the terminals appear, in a counterclockwise traversal of the outer face, in the order $s_1, s_2, \ldots, s_k, t_{\pi(1)}, t_{\pi(2)}, \ldots, t_{\pi(k)}$ for some permutation π of the numbers $1, 2, \ldots, k$. See Figure 6.2 for an example.

The setting has the following important property. Every path $P \in \mathcal{P}_i$ separates the planar embedding of the graph into two parts. The part *above* P is the set in the plane enclosed by the curve formed by the boundary of the outer face between t_i and s_i (in counterclockwise order) and by path P (from s_i to t_i). The part *below* P is the set in the plane enclosed by the curve formed by the boundary of the outer face between s_i and t_i (in counterclockwise order) and by path P (from t_i to s_i). In the following we say that a point/path/vertex/etc. *lies above* P if it lies in the part above P. We similarly define *to lie below* P. Observe that both sets are compact and closed. They share only path P and otherwise are disjoint. Therefore any path P' that lies above P and is disjoint from P is also disjoint from any path P'' that lies below P.

As we will show in the following, the separating cut imposes an order structure on the set of all alternative paths $\mathcal{R}$. A *strict partial order* over a set is a binary relation $<$ that is irreflexive, transitive, and hence asymmetric. Two elements a and b of the set are *comparable* if either $a < b$ or $b < a$. The set together with the partial order is



(a)



(b)

Figure 6.2: An instance of VDP-SEP, i.e., all terminals lie on the outer face of G and there is a separating cut in G. (a) The border of the outer face is depicted in bold. (b) An s_3-t_3 path p (dashed bold line), the part above p (dark shaded area), and the part below p (light shaded area).

called a *partially ordered set* or a *poset*, for short. In the following, we define a binary relation $<_{\mathcal{R}}$ on the set of all alternative paths and show that $(\mathcal{R}, <_{\mathcal{R}})$ is a poset: Let $P_i <_{\mathcal{R}} P_j$ if path $P_i \in \mathcal{R}$ is above $P_j \in \mathcal{R}$.

Theorem 6.8. *For every instance with a separating cut the set of all alternative paths $\mathcal{R}$ forms a poset $(\mathcal{R}, <_{\mathcal{R}})$. Furthermore, P_i and P_j are comparable in the poset if and only if they are vertex disjoint.*

Proof. For any two paths $P_i, P_j \in \mathcal{R}$, if $P_i <_{\mathcal{R}} P_j$ (i.e., P_i is above P_j) then P_i does not intersect P_j and $i < j$ (recall that the terminals appear on the boundary in sorted order $s_1, \ldots, s_k$). Conversely, if two paths P_i and P_j are vertex disjoint we have either $P_i <_{\mathcal{R}} P_j$ or $P_j <_{\mathcal{R}} P_i$ (i.e., P_i and P_j are comparable). It is easy to see that $<_{\mathcal{R}}$ is irreflexive, since no path is disjoint from itself, and transitive, due to the separating cut. Hence, $(\mathcal{R}, <_{\mathcal{R}})$ is a poset. □

A *chain* is a totally ordered subset of a poset (i.e., any two elements of the subset are comparable).

Theorem 6.9. M-VDP-SEP *reduces to finding a maximum chain in* $(\mathcal{R}, <_{\mathcal{R}})$.

Proof. Let $\mathcal{I}$ be an instance of M-VDP-SEP. A feasible solution of $\mathcal{I}$ is a set $S \subseteq \mathcal{R}$ of vertex disjoint paths. The paths of S are pairwise comparable, so S is a chain in $(\mathcal{R}, <_{\mathcal{R}})$. It is easy to see that there is a bijection between optimal solutions of $\mathcal{I}$ and maximum chains in $(\mathcal{R}, <_{\mathcal{R}})$. □

It is well known that computing a maximum chain of a poset can be reduced to computing a longest path in the corresponding directed acyclic graph (having one node per element of the poset and a directed arc (u, v) if and only if $u < v$ in the poset). The directed acyclic graph can be constructed from the instance of M-VDP-SEP in $O((kp)^2|V|)$ time. Computing a longest path in that graph takes linear time in the size of the graph, i.e. $O((kp)^2)$.

Corollary 6.10. M-VDP-SEP *and hence* D-VDP-SEP *can be solved in polynomial time.*

Let us briefly point out some connections to graph theory. Namely, the poset of Theorem 6.8 gives rise to a comparability graph, which belongs to the class of perfect graphs, see, e.g., [46]. The complement of that graph, a cocomparability graph, is just the intersection graph of the paths of $\mathcal{R}$. It can be shown that the class of intersection graphs corresponding to the instances of VDP-SEP is exactly the class of cocomparability graphs, using the concept of function diagrams, see, e.g., [47].

6.4 R-VDP : Routing in Rounds

In this section we consider variants of the routing-in-rounds problem. An instance is a triple $(G, T, \mathcal{P})$, where G is a plane graph, T is a collection of k terminal pairs $\{s_i, t_i\}$, $i = 1, 2, \ldots, k$, and $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$, where $\mathcal{P}_i$ is a set of s_i-t_i paths. A solution is a set of paths $S = \{P_i\}_{i=1\ldots k}$, $P_i \in \mathcal{P}_i$, along with a proper coloring $c : S \to \{1, \ldots, r\}$, i.e., each path $P_i \in S$ is assigned a color $c(i) \in \mathbb{N}$,

such that for all $P_i, P_j \in S$, if $P_i \cap P_j \neq \emptyset$ then $c(P_i) \neq c(P_j)$. Intuitively, the colors correspond to rounds, where paths assigned to the same round are required to be vertex disjoint.

In the following, we show that already R-VDP-SOR (terminals sorted on the outer face) is APX-complete for any $p \geq 2$. Further we show that R-VDP-SEP (there is a separating cut) with $p = 1$ can be solved efficiently, and present a p-approximation algorithm for the case of $p \geq 2$.

The more general case R-VDP-OUT with $p = 1$, i.e., where for each terminal pair a path has already been chosen, is polynomially equivalent to coloring an *outerstring* graph, see Section 6.3.2. The latter problem is already NP-complete for circle graphs [42], a subclass of outerstring graphs.

6.4.1 R-VDP-SOR: Terminals Sorted on the Outer Face

Theorem 6.11. R-VDP-SOR *for any $p \geq 2$ is APX-complete.*

Proof. We first prove APX-hardness by reduction from SETCOVER, which is defined as follows. Given a collection $\mathcal{C}$ of subsets of a ground set U, the SETCOVER problem asks for a collection $\mathcal{C}' \subseteq \mathcal{C}$, such that each $u_i \in U$ belongs to at least one member of $\mathcal{C}'$ and $|\mathcal{C}'|$ is minimized, see [41, SP5]. The SETCOVER problem is APX-complete [70] when the number of occurrences of an element in sets of $\mathcal{C}$ is bounded from above by any constant $B \geq 2$.

In the reduction, as illustrated in Figure 6.3, we transform any instance of SETCOVER as follows. Every element $u_i \in U$ corresponds to one terminal pair $\{s_i, t_i\}$. We let these terminal pairs be drawn one below another in the plane graph we construct, the order is arbitrary. Each occurrence of u_i in a set $C_j \in \mathcal{C}$ corresponds to one s_i-t_i path. Except for a peak at a paticular location, an s_i-t_i path follows a straight line from s_i to t_i. The position of the peak represents the set C_j in which u_i occurs for that particular occurrence. For two different elements u_i, u_j occurring in the same set C_l, we let the corresponding paths be non-intersecting, by aligning their peaks together in the position of C_l. If two elements occur in two different sets, the corresponding paths intersect because their peaks are not aligned. The peaks follow the shape shown in Figure 6.3. By this construction, two

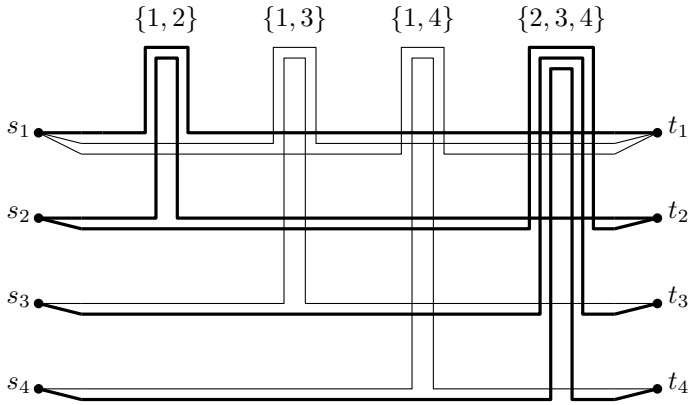


Figure 6.3: Reduction from set cover. Every element $u_i \in U$ is transformed into a terminal pair $\{s_i, t_i\}$. Each occurrence of an element u_i in a subset $C_j \in C$ is transformed into an s_i-t_i path, such that two paths are disjoint if and only if they represent elements of the same set. Example with $U = \{1, 2, 3, 4\}$, and $C = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3, 4\}\}$. The chosen paths are drawn bold and correspond to sets $\{1, 2\}$ and $\{2, 3, 4\}$. Note that the terminal pair $\{s_2, t_2\}$ is covered twice.

paths of different terminal pairs can be scheduled in the same round if and only if the corresponding elements belong to the same set in $\mathcal{C}$. The minimum number of rounds needed to schedule all terminal pairs equals $|C'|$.

It is easy to see that the above reduction is approximation preserving. By the p-approximation algorithm given in Section 6.4.3, the claim follows. □

6.4.2 R-VDP-SEP: Separating Cut, $p = 1$

A consequence of Theorem 6.8 is that once a path is selected for each terminal pair, the assignment of paths to a minimum number of rounds is solvable in polynomial time as follows: We can see R-VDP-SEP with $p = 1$ as the problem of covering a poset with a minimum number of chains. Recall that a *chain* of a poset $(\mathcal{R}, <_{\mathcal{R}})$ is a subset of totally ordered elements of $\mathcal{R}$. Thus, a chain of the

poset $(\mathcal{R}, <_{\mathcal{R}})$ corresponds to paths that can be scheduled in the same round. A *chain cover* of the poset is a set of chains such that every element of $\mathcal{R}$ is in (at least) one chain. Since $p = 1$, all paths need to be scheduled, and a chain cover of minimum cardinality corresponds to routing of the paths in minimum number of rounds. The problem of covering a poset with a minimum number of chains has been well studied (see for example the characterization of solutions known as Dilworth's theorem [23]), and can be solved in polynomial time by computing a maximum matching in a related bipartite graph [32].

Corollary 6.12. R-VDP-SEP *with* $p = 1$ *can be solved in polynomial time.*

6.4.3 R-VDP-SEP: Separating Cut, $p \geq 2$

Since R-VDP-SEP is APX-hard for $p \geq 2$, the question arises how well one can approximate variants of the routing-in-rounds problem. For the SETCOVER problem it is known that it cannot be approximated below a threshold of $(1 - o(1)) \ln |U|$, where U is the ground set to be covered, unless NP has slightly superpolynomial time algorithms [24]. Thus, the greedy algorithm for set cover is essentially the best one can hope for, see, e.g., [81]. There is, however, a B-approximation algorithm for SETCOVER if each element is covered by at most $B \geq 2$ sets [51]. In the following, we give a p-approximation algorithm for R-VDP-SEP.

Let $(\mathcal{R}, <_{\mathcal{R}})$ be the poset of an input instance of R-VDP-SEP as defined in Section 6.3.3. An *antichain* of a poset is a subset in which no two elements are comparable. Hence, an antichain of $(\mathcal{R}, <_{\mathcal{R}})$ is a set of mutually intersecting paths. Let $\mathcal{A}(\mathcal{R}, <_{\mathcal{R}})$ denote the set of all maximal antichains of $(\mathcal{R}, <_{\mathcal{R}})$. Consider the following integer program to calculate the minimum number of rounds r needed

to schedule all terminal pairs of R-VDP-SEP instance:

$$\begin{aligned}
\text{(IP)} \quad & \min && r && && \text{(6.1a)}\\
& \text{s.t.} \\
& && \sum_{P_j \in \mathcal{P}_i} x_{ij} = 1 && \forall\, \{s_i, t_i\} \in T && \text{(6.1b)}\\
& && \sum_{\substack{i,j \\ P_j \in \mathcal{P}_i \cap A}} x_{ij} \leq r && \forall\, A \in \mathcal{A}(\mathcal{R}, <_{\mathcal{R}}) && \text{(6.1c)}\\
& && x_{ij} \in \{0,1\} && && \text{(6.1d)}
\end{aligned}$$

The binary variables x_{ij} denote whether the j'th path from set $\mathcal{P}_i$ is selected. Constraints (6.1b) require that for each terminal pair $\{s_i, t_i\}$, $i = 1, \ldots, k$, exactly one path in the corresponding set $\mathcal{P}_i$ is chosen. Constraints (6.1c) require for each (maximal) antichain A that there are at least as many rounds r as the number of paths chosen in A (which are mutually intersecting). Note that there may be exponentially many constraints of type (6.1c).

Lemma 6.13. *The value r^* of an optimal solution to (IP) equals the minimum number of rounds R needed to schedule a corresponding instance of* R-VDP-SEP.

Proof. Consider an optimal solution of (IP). The variables x^*_{ij} represent the choice of paths in this solution. Let $(\mathcal{R}', <_{\mathcal{R}'})$ be the poset induced by this choice of paths, i.e., $\mathcal{R}' := \{P_j \in \mathcal{P}_i \mid x^*_{ij} = 1\}$ and $<_{\mathcal{R}'} := <_{\mathcal{R}} \cap\, \mathcal{R}' \times \mathcal{R}'$.

Since r^* is minimal in an optimal solution, Constraint (6.1c) holds with equality for some $A^* \in \mathcal{A}(\mathcal{R}, <_{\mathcal{R}})$. Clearly, for every antichain A' of $(\mathcal{R}', <_{\mathcal{R}'})$ there is an antichain A of $\mathcal{A}(\mathcal{R}, <_{\mathcal{R}})$ such that $A' \subseteq A$. Therefore, $r^* = \max\{|A'| \mid A' \in \mathcal{A}'(\mathcal{R}', <_{\mathcal{R}'})\}$, i.e., r^* equals the size of a maximum antichain of $(\mathcal{R}', <_{\mathcal{R}'})$. By Equation (6.1b) the set $\mathcal{R}'$ contains exactly one path per terminal pair, i.e., $|\mathcal{R}' \cap \mathcal{P}_i| = 1$ for each $i = 1, \ldots, k$. Hence, we may apply Dilworth's theorem as described in Section 6.4.2 (R-VDP-SEP with $p = 1$). Namely, the minimum number of chains (rounds) needed to cover all paths in $\mathcal{R}'$ equals the size of a maximum antichain of $(\mathcal{R}', <_{\mathcal{R}'})$. Hence, the lemma follows. □

We note that the actual colors $c(i)$ for each chosen path $P_j \in \mathcal{P}_i$ can be found in polynomial time by Corollary 6.12, where each chain of the minimum chain cover corresponds to a color class.

Denote by (LP) the linear relaxation of (IP), and by (LP') the linear program (6.1a)-(6.1b). Note that constraints (6.1c) can be separated in polynomial time by the weighted Dilworth theorem [40]. That is, given a feasible solution to (LP'), we can find (in polynomial time) a violated constraint of (6.1c), if there is one, by finding a maximum weighted antichain in the poset induced by (fractional values of) the x_{ij}. Namely, we compute the size of a maximum weighted antichain for the set $\mathcal{R}'' := \{P_j \in \mathcal{P}_i \mid x_{ij} > 0, i = 1 \ldots, k\}$, with weights given by the values of the variables x_{ij}. If this antichain does not violate (6.1c), no other antichain does. By the polynomial equivalence of optimization and separation, see [49], (LP) can be solved in polynomial time.

We obtain the desired approximation by rounding any fractional values of x_{ij}, i.e., the solution of the (LP). For each terminal pair $\{s_i, t_i\}$, we choose an $x_{ij'}$ with maximum value, denoted by $\overline{x}_i$, and round it to 1. We round the remaining x_{ik}, $k \neq j'$, to 0.

Theorem 6.14. R-VDP-SEP *with at most p alternative paths per terminal pair can be approximated within a factor of p.*

Proof. Let x^* be an optimal solution to (LP) with objective value r^*. Denote by R the value of an optimal solution to R-VDP-SEP. Clearly, $R \geq r^*$. The rounded values are feasible with respect to equations (6.1b). Given that there are at most p paths per terminal pair, we have $\overline{x}_i \geq 1/p$ for all terminal pairs $\{s_i, t_i\} \in T$. Hence, each x_{ij} is rounded up by a factor of at most p. Therefore, equations (6.1c) are satisfied for a right hand side of $r^* \cdot p$. Hence, the objective value of the returned solution is at most $p \cdot R$. □

R-VDP-SEP With at Most Two Paths per Round

The reduction from SETCOVER in Section 6.4.1 suggests an interesting question. We have seen that an instance of SETCOVER can be expressed as R-VDP-SEP. It is known that SETCOVER is solvable in polynomial time if every set with which we are to cover the universe of elements has size at most 2 as it reduces to the edge cover problem [41, 67]. In our reduction this would correspond to the setting where

each path is compatible with at most one other path (i.e., every path intersects all other paths, or all other paths but one). Indeed, this case of R-VDP-SEP is solvable in polynomial time:

Theorem 6.15. R-VDP-SEP *(with arbitrary p) where each path $P \in \mathcal{R}$ is vertex disjoint from at most one other path $P' \in \mathcal{R}$ is solvable in polynomial time.*

Proof. Consider the graph $G = (V, E)$, where each terminal pair $\{s_i, t_i\}$ is represented by a vertex $v_i \in V$, and there is an edge $(v_i, v_j) \in E$ if and only if the corresponding terminal pairs can be scheduled together, i.e., there is a path $P \in \mathcal{P}_i$ and a path $P' \in \mathcal{P}_j$ that are vertex disjoint.

It is easy to see that the minimum number of rounds R corresponds to the size of a maximum matching M in G, plus the number of unmatched vertices u: by assumption, at most two terminal pairs can be scheduled in one round. Hence, $|M|$ is the maximal number of rounds in which two terminal pairs can be scheduled simultaneously. It follows that $R = |M| + u$. □

Chapter 7

MIS in Outersegment Graphs

7.1 Introduction

In this chapter we study the problem of computing a maximum independent set (MIS) in intersection graphs of segments lying inside a disk that are either horizontally or vertically aligned and have one endpoint attached to the boundary of the disk. The problem of computing a MIS in various classes of intersection graphs has been intensively studied in the literature. For an extensive survey on many graph classes, refer to [13]. Despite the numerous efforts, the problem is by far not solved or fully understood. This chapter adds to these efforts by presenting a polynomial-time algorithm for computing a MIS in a specific class of intersection graphs.

Motivated by general interest in the field of computational geometry and graph theory, intersection graphs of curves in the plane have received considerable attention in the literature, e.g., see [35, 36, 37, 57, 58, 59, 69, 74]. A graph is a *string graph* if each vertex can be represented by a *string*, i.e., a curve in the Euclidean plane, such that there is an edge connecting two vertices if and only if the corresponding strings intersect. A set of strings representing the vertices is called a *representation* of the graph.

Most of the classical NP-hard optimization problems on graphs

(such as finding a maximum clique, a maximum independent set, a minimum vertex cover, a minimum dominating set, or a minimum coloring) remain NP-hard for string graphs even if the representation is given [55, 58, 60, 80]. The problem of recognizing string graphs, i.e., deciding whether a given graph is a string graph, is NP-hard, too [58]. Finding a MIS remains NP-hard even for the yet narrower class of *segment graphs*, which are the intersection graphs of straight line segments in the plane [60]. If every segment of the representation of a segment graph follows one of d directions, we say that the graph is a *d-direction segment graph*. It has been shown that the problem of computing a MIS in d-direction segment graphs is NP-hard for every $d \geq 2$ [60], whereas it is solvable in polynomial time for $d = 1$, since a 1-direction segment graph is an interval graph.

Considerably less is known about the problem of computing a MIS in string graphs if we restrict the strings to lie entirely inside a disk and to have one endpoint on the boundary of the disk. Such a string graph is called an *outerstring graph*. While finding a maximum clique is NP-hard in outerstring graphs [66], the complexity of finding a MIS in outerstring graphs is, to the best of our knowledge, an open problem.

We call an outerstring graph an *outersegment* graph if it has a representation where every string is a straight line segment. If further every segment of the representation follows one of d fixed directions, we say that the graph is a *k-direction outersegment graph*.

In this chapter we present a polynomial-time algorithm for the problem of finding a MIS in a 2-direction-outersegment graph if a representation of the outersegment graph is given where each segment is either horizontally or vertically aligned. We refer to this computational problem as MIS-ORTH-OSEG and to the class of intersection graphs as ORTH-OSEG. The main ingredient of our solution is a dynamic-programming algorithm that solves the problem on restricted instances where no vertical segment attached to the upper half of the disk appears. Then, by a careful guessing of few segments of an optimal solution, we can decompose the original problem into four restricted subproblems where we can apply the dynamic programming algorithm.

Let us remark that the restriction to graphs of ORTH-OSEG still allows for chordless cycles of length 5 (refer to [13] for terminology). Hence, outersegment graphs are not perfect.

We leave the complexity of computing a MIS in d-direction outersegment graphs open for $d \geq 3$. To the best of our knowledge, the complexity of recognizing ORTH-OSEG graphs is open, too.

7.1.1 Notation and Definitions

An instance of MIS-ORTH-OSEG is a set $\mathcal{I}$ of straight line segments in the plane lying in a disk $\mathcal{D}$ such that each segment s is either horizontally or vertically aligned and has at least one endpoint on the boundary of $\mathcal{D}$. We call this endpoint the *disk-endpoint* of s. We assume w.l.o.g. that the other endpoint of s does not lie on the boundary of $\mathcal{D}$. We call this endpoint the *free-endpoint* of s.

To facilitate our discussion, we assume w.l.o.g. that the center of $\mathcal{D}$ is aligned with the origin of a Cartesian coordinate system. Furthermore, every segment $s \in \mathcal{I}$ is either *horizontal* (i.e., parallel with the x-axis) or *vertical* (i.e., parallel with the y-axis). We assume w.l.o.g. that no segment lies on the x-axis or on the y-axis. Thus, each segment is either a *left-, right-, top-* or *bottom-* segment, depending on the location of its disk-endpoint: a horizontal segment is a left-segment (right-segment), if its disk-endpoint has a negative (positive) x-coordinate; a vertical segment is a top- (bottom-) segment, if its disk-endpoint has a positive (negative) y-coordinate. We denote the set of left-, right-, top-, and bottom-segments as $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{B}$, respectively. These sets form a partition of $\mathcal{I}$.

As our goal is to compute a MIS in the intersection graph of $\mathcal{I}$, we assume w.l.o.g. that no two segments in $\mathcal{L}$ have the same disk-endpoint: observe that no MIS can contain more than one such segment; thus we can preprocess the input by keeping in $\mathcal{L}$ the shortest segment of all segments with the same disk-endpoint. Similarly, we assume the same about segments in $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{B}$. Thus, the segments within one set of the partition do not intersect (and so form an independent set in the underlying intersection graph). Note, however, that a horizontal segment from $\mathcal{L}$ may intersect with one from $\mathcal{R}$. Similarly, a vertical segment from $\mathcal{T}$ may intersect with one from $\mathcal{B}$. We call an instance *bipartite* if at least two of the sets of the partition $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{B}$ are empty. Clearly, the intersection graph of a bipartite instance is a bipartite graph, for which the problem of finding a MIS can be solved in polynomial time [77]. We call an instance *tripartite* if one of the sets of the partition $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{B}$ is empty. We refer to the

version of MIS-ORTH-OSEG that is restricted to tripartite instances as *tripartite* MIS-ORTH-OSEG.

We distinguish between two different locations of the vertical segments: a *western vertical segment* lies to the left of the y-axis, and an *eastern vertical segment* lies to the right of the y-axis. Similarly, we distinguish two different locations of the horizontal segments: a *northern horizontal segment* lies above the x-axis, and a *southern horizontal segment* lies below the x-axis.

Finally, for a region $X \subset \mathcal{D}$ and a set of segments S, we denote by $S[X]$ the segments of S contained entirely in X.

Outline In Section 7.2 we present a polynomial-time algorithm for tripartite MIS-ORTH-OSEG, i.e., the restricted version of MIS-ORTH-OSEG where one set of the partition $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{B}$ is empty. Based on this, we present a polynomial-time algorithm for MIS-ORTH-OSEG in Section 7.3.

7.1.2 Summary of Results

We present a polynomial-time algorithm for the problem of computing a maximum independent set in outersegment graphs where every segment is either horizontally or vertically aligned. We assume that a geometric representation of the graph is given as input.

7.2 Solving Tripartite MIS-ORTH-OSEG

In this section we consider tripartite instances of MIS-ORTH-OSEG, i.e., instances for which one of the sets $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{B}$ is empty. Without loss of generality, we will assume that $\mathcal{T} = \emptyset$, i.e., that there is no top segment. In the following we present a polynomial-time algorithm that finds a maximum independent set in any such restricted instance. We will observe the existence of a certain *decomposition of every solution* into solutions of independent subproblems, where every subproblem is a bipartite instance of MIS-ORTH-OSEG. We describe this decomposition in the next section, before presenting the actual algorithm which is based on the dynamic programming technique.

7.2.1 Structure of an Optimal Solution

In the following we show that an optimal solution for tripartite MIS-ORTH-OSEG is the disjoint union of optimal solutions for a certain set of subproblems. Let $\mathcal{I}$ be an instance of tripartite MIS-ORTH-OSEG. Assume that we are given an optimal solution OPT for $\mathcal{I}$ together with a partition of the disk $\mathcal{D}$ into regions $\mathcal{D}_1, \ldots, \mathcal{D}_r$ such that each segment of OPT lies entirely inside one of these regions. For $i = 1, \ldots, r$, recall that $\mathcal{I}[\mathcal{D}_i]$ and $\text{OPT}[\mathcal{D}_i]$ denote the set of segments of the instance $\mathcal{I}$ and OPT, respectively, that lie completely within region $\mathcal{D}_i$. Note that while $\text{OPT} = \bigcup_{i=1}^{r} \text{OPT}[\mathcal{D}_i]$, it holds that $\mathcal{I} \supsetneq \bigcup_{i=1}^{r} \mathcal{I}[\mathcal{D}_i]$ if there is a segment of $\mathcal{I}$ that lies in more than one of the regions. Denote by $\text{MIS}[S]$ a maximum independent set for a set S of segments. For a region $\mathcal{D}_i$, we abbreviate $\text{MIS}[\mathcal{I}[\mathcal{D}_i]]$ to $\text{MIS}[\mathcal{D}_i]$. Clearly, it follows that $|\text{OPT}[\mathcal{D}_i]| = |\text{MIS}[\mathcal{D}_i]|$ for all $i = 1, \ldots, r$.

Hence, if we can find such a partition in polynomial time such that for all $i = 1, \ldots, r$, a $\text{MIS}[\mathcal{D}_i]$ can be computed in polynomial time, then we can solve tripartite MIS-ORTH-OSEG in polynomial time. We show that such a partition always exists before presenting our algorithm in the next section. The argument is based on the structure of an (unknown) optimal solution OPT. We show that a particular traversal of the bottom segments of OPT yields the desired partition of $\mathcal{D}$. We say that a bottom segment s_i *towers above* a bottom segment s_j if the y-coordinate of the free-endpoint of s_i is greater than that of s_j. W.l.o.g. we assume that the free-endpoints lie in general position and hence that for each region, there is at most one segment towering above all other segments.

Lemma 7.1. *Given an instance $\mathcal{I}$ of tripartite* MIS-ORTH-OSEG *and an optimum solution OPT, there exists a partition $\mathcal{D}_1, \ldots, \mathcal{D}_r$ of the disk $\mathcal{D}$ into regions such that $|OPT| = \sum_{i=1}^{r} |\text{MIS}[\mathcal{D}_i]|$ and a $\text{MIS}[\mathcal{D}_i]$ can be computed in polynomial time for each $i = 1, \ldots, r$.*

Proof. Given an optimal solution OPT, we partition the disk recursively as follows, see Figure 7.1 for an example. At each step i of the recursion, there is an *unprocessed* region U_i of the disk for which $\mathcal{I}[U_i]$ is tripartite. Let $U_0 := \mathcal{D}$ be the initial region. At each step $i \geq 1$, let s_i be the bottom segment in U_{i-1} that towers above all other bottom segments in U_{i-1}. Segment s_i naturally divides U_{i-1} into three regions, namely A_i, B_i, and U_i, as follows.

A_i is the region (strictly) above s_i. Note that $\mathcal{I}[A_i]$ may contain bottom segments if it does not lie completely in the northern half of $\mathcal{D}$, and thus be tripartite. In this case, however, it must hold that a MIS for $\mathcal{I}[A_i] \setminus \mathcal{B}$ is also a MIS for $\mathcal{I}[A_i]$, since by choice of s_i, OPT$[A_i]$ does not contain a bottom segment. Hence, a MIS$[A_i]$ can be computed in polynomial time, as $\mathcal{I}[A_i] \setminus \mathcal{B}$ is bipartite.

B_i and U_i are the regions to either side of s_i (and below A_i), where B_i includes s_i and U_i contains the lowest point of the disk. (For example, if s_i is a western bottom segment, B_i is to the left of s_i.) Because B_i does not contain the lowest point of the disk, it either lies completely in the left or completely in the right half of the disk. Hence, no two horizontal segments of $\mathcal{I}[B_i]$ can intersect, as they are all either left or right segments. Thus, $\mathcal{I}[B_i]$ is bipartite as well.

The recursion continues by partitioning U_i if OPT$[U_i]$ contains a bottom segment, and stops otherwise. Let U_ℓ denote the region at which the recursion stops. Similar to the argument for A_i above, it suffices to compute a MIS$[\mathcal{I}[U_\ell] \setminus \mathcal{B}]$, as OPT$[U_\ell]$ consists of horizontal segments only. As $\mathcal{I}[U_\ell] \setminus \mathcal{B}$ is bipartite, a MIS can be computed in polynomial time.

By the choice of s_i, the regions A_i, B_i, $i = 1, \ldots, \ell$ and U_ℓ are a partition of $\mathcal{D}$ such that each segment of OPT lies completely within one of these regions. Hence,

$$|\text{OPT}| = \sum_{i=1}^{\ell} |\text{MIS}[A_i]| + |\text{MIS}[B_i]| + |\text{MIS}[U_\ell]|$$

must hold. This completes the proof. □

In the following, we call A_i the part *above* s_i in U_{i-1}, B_i the part *behind* s_i in U_{i-1}, and U_i the *unprocessed* part of U_{i-1} by s_i.

7.2.2 Algorithm for Tripartite MIS-ORTH-OSEG

From the above discussion we know that there exists a sequence of bottom segments $s_1, \ldots, s_\ell$ that yields a partition of $\mathcal{D}$ into regions such that a MIS for $\mathcal{I}$ can be computed in polynomial time by independently computing a MIS for a set of bipartite subproblems induced by these regions. The partition of $\mathcal{D}$ is based on the structure

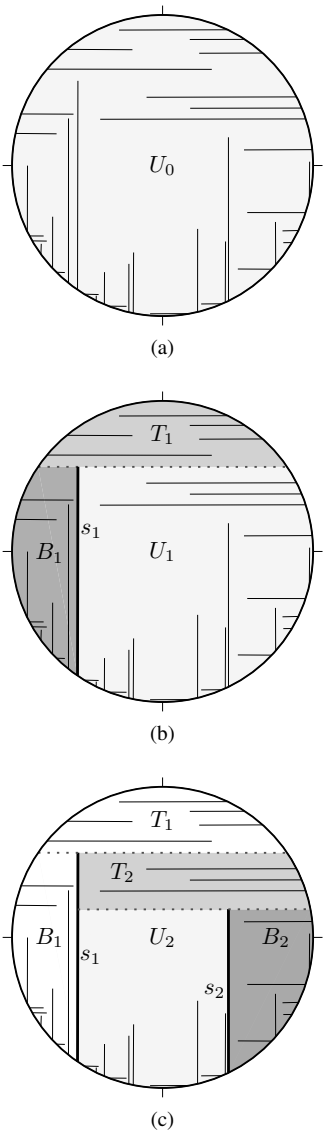


Figure 7.1: All three pictures depict the same set of independent segments of an optimal solution to a tripartite instance of MIS-ORTH-OSEG. (a) Initially, the unprocessed region U_0 is the whole disk $\mathcal{D}$. (b) The bottom-segment s_1 of an optimal solution OPT partitions U_0 into three regions A_1, B_1, and U_1. (c) Recursively, s_2, which towers above all bottom-segments in U_1, partitions U_1 into three regions A_2, B_2, and U_2.

of an optimal solution which, of course, is unknown. Next, we develop a dynamic programming approach that allows us to find such a partition.

To develop the algorithm, we need a few more definitions. Let $s_L(i)$ and $s_R(i)$ denote the last visited western and eastern bottom-segment, respectively, after step $i = 1, \ldots, \ell$, of the recursion in the proof of Lemma 7.1. In the following we will use two *phantom boundary* segments $s_{-\infty}$ and s_{∞} to allow $s_L(i)$ and $s_R(i)$ to be always defined: we denote by $s_{-\infty}$ the infinite vertical segment defined by the equation $x = -\infty$ (i.e, a line), and by ℓ_{∞} the infinite vertical segment $x = +\infty$. We set $s_L(0) = s_{-\infty}$ and $s_R(0) = s_{\infty}$.

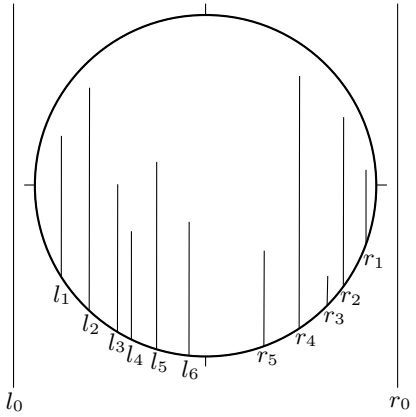
Observe that with these definitions, the region U_i, $i = 0, \ldots, \ell$, is defined by $s_L(i)$ and $s_R(i)$: U_i is the region of $\mathcal{D}$ to the right of $s_L(i)$ and to the left of $s_R(i)$ and below the free-endpoints of both $s_L(i)$ and $s_R(i)$. The regions A_{i+1} and B_{i+1} can thus be defined by U_i and the $(i+1)$-th visited segment, i.e., by $s_L(i)$, $s_R(i)$ and by either $s_L(i+1)$ or $s_R(i+1)$.

Let $l_1, \ldots, l_{n_l}$ be the western bottom-segments sorted by x-coordinate in increasing order, and let $r_1, \ldots, r_{n_r}$ be the eastern bottom-segments sorted by x-coordinate in decreasing order. Further, let l_0 be the phantom segment $s_{-\infty}$ and let r_0 be the phantom segment s_{∞}. See Figure 7.2 for illustration.

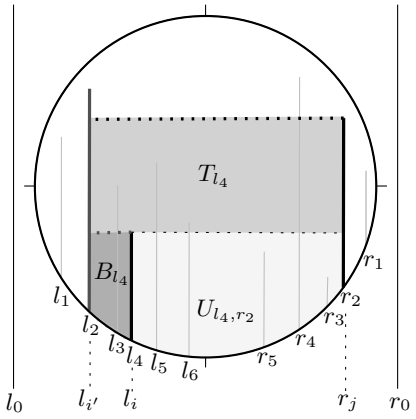
For a segment l_i and a segment r_j we define by U_{l_i,r_j} the unprocessed region that we would obtain by the recursion if l_i and r_j were the last visited western and eastern bottom-segment, respectively. Thus, this is the region between segments l_i and r_j and below the free-endpoints of l_i and r_j.

We will compute the table $\mathrm{T}[l_i, r_j]$ for every $i = 0, \ldots, n_l$ and every $j = 0, \ldots, n_r$, where $\mathrm{T}[l_i, r_j]$ is the maximum number of non-intersecting segments in the subproblem defined by the segments lying completely inside $\mathcal{D} \setminus U_{l_i,r_j}$ where segments l_i and r_j are required to be part of the solution (of the subproblem).

Clearly, if we have such a table at hand, we can compute the optimal number of non-intersecting segments of the whole instance: consider the recursion in the proof of Lemma 7.1 on an (unknown) optimal solution OPT. Let $l_i \in \mathrm{OPT}$ be the last western and $r_j \in \mathrm{OPT}$ be the last eastern bottom-segment encountered in the sequence $s_1, \ldots, s_\ell$ of the recursion. By Lemma 7.1, a $\mathrm{MIS}[U_{l_i,r_j}]$ can be



(a)



(b)

Figure 7.2: Both figures depict the bottom segments of a (tripartite) instance of MIS-ORTH-OSEG. (a) The western bottom-segments $l_0, \ldots, l_{n_l}$ are ordered from left to right (here, $n_l = 6$), and the eastern bottom-segments $r_0, \ldots, r_{n_r}$ are ordered from right to left (here, $n_r = 5$). (b) Computing $\mathrm{T}[l_i, r_j]$ as $\mathrm{T}[l_{i'}, r_j] + |\mathrm{MIS}[A_{l_i}]| + |\mathrm{MIS}[B_{l_i}]|$ for $i = 4$, $j = 2$ and $i' = 2$.

computed in polynomial time (because it suffices to compute a MIS for the bipartite instance $\mathcal{I}[U_{l_i,r_j}] \setminus \mathcal{B}$). Hence, $|\text{OPT}| = \text{T}[l_i, r_j] + |\text{MIS}[U_{l_i,r_j}]|$ can be computed in polynomial time.

As we do not know OPT, our algorithm tries all pairs $l_i, r_j \in \mathcal{B}$, and outputs the maximum of the computed values $\text{T}[l_i, r_j] + |\text{MIS}[U_{l_i,r_j}]|$ over all i, j. The solution itself can be computed using standard book-keeping techniques.

We now show how to compute the entries of the table $\text{T}[\cdot,\cdot]$. Again, for a pair $s, t \in \mathcal{B}$, we say that s *towers above* t if the free-endpoint of s has a greater y-coordinate than that of t.

We set $\text{T}[l_0, r_0] = 0$. Then, for every $i = 0, \ldots, n_l$ and every $j = 0, \ldots, n_r$, we need to distinguish the following cases in order to compute the value of entry $\text{T}[l_i, r_j]$. Namely, for the case that r_j towers above l_i, we compute

$$\text{T}[l_i, r_j] = \max_{\substack{i'<i \\ l_{i'} \text{ towers above } l_i \text{ and } r_j}} \{\text{T}[l_{i'}, r_j] + |\text{MIS}[A_{l_i}]| + |\text{MIS}[B_{l_i}]|\}, \tag{7.1}$$

and otherwise (if l_i towers above r_j), we compute

$$\text{T}[l_i, r_j] = \max_{\substack{j'<j \\ r_{j'} \text{ towers above } l_i \text{ and } r_j}} \{\text{T}[l_i, r_{j'}] + |\text{MIS}[A_{r_j}]| + |\text{MIS}[B_{r_j}]|\}. \tag{7.2}$$

As in the proof of Lemma 7.1, A_{l_i} is the region of $\mathcal{D}$ between segments $l_{i'}$ and r_j and above the free-endpoint of l_i and not above the free-endpoints of $l_{i'}$ and r_j; B_{l_i} is the region of $\mathcal{D}$ below A_{l_i} and between $l_{i'}$ and l_i, including l_i but not $l_{i'}$. The regions A_{r_j} and B_{r_j} are defined symmetrically. By Lemma 7.1, the cardinalities of $|\text{MIS}[A_s]|$ and $|\text{MIS}[B_s]|$, $s \in \{l_i, r_j\}$ can be computed in polynomial time.

Theorem 7.2. *The table entry $T[l_i, r_j]$, $i = 0, \ldots, n_l$, $j = 0, \ldots, n_r$, contains the size of an optimal solution of an instance $\mathcal{I}[\mathcal{D} \setminus U_{l_i,r_j}]$ further restricted to contain the segments l_i and r_j.*

Proof. To prove the theorem we need to show that $\text{T}[\cdot,\cdot]$ indeed has the recursive property of Equations (7.1) and (7.2). This, however, follows directly from the existence of a decomposition as described in Section 7.2.1: The recursive computation of $\text{T}[l_i, r_j]$ takes the last visited segment x (x is the "smaller" segment of l_i or r_j, i.e., the one that is not towering above the other) and finds the segment that is the

predecessor of x in the sequence $s_1, \ldots, s_\ell$ of bottom segments of an unknown optimal solution OPT. This predecessor of x naturally defines, together with x, the region A_x above x and the region B_x behind x, just as in the recursion of Lemma 7.1. The correctness of the recursive definition of T then directly follows from Lemma 7.1. □

Corollary 7.3. *There is a polynomial-time algorithm for tripartite instances of* MIS-ORTH-OSEG.

7.3 Decomposing MIS-ORTH-OSEG

In this section we provide a polynomial time algorithm for the general setting. We show how to decompose an arbitrary instance of MIS-ORTH-OSEG into few tripartite instances of MIS-ORTH-OSEG. The decomposition we describe can be computed in polynomial time. Combined with the polynomial time algorithm for tripartite MIS-ORTH-OSEG presented in the previous section, it yields a polynomial time algorithm for MIS-ORTH-OSEG. The decomposition is determined by a constant number of segments in an optimal solution. Since we do not know these segments, we have to perform an exhaustive search, namely by enumerating through all sets of segments whose cardinality is bounded by a constant.

We will use the following notation. A vertical *overlap* is a pair of vertical segments that cannot be separated by a horizontal line. If a vertical overlap consists of western segments only, i.e., if it lies entirely to the left of the y-axis, we call it a *left overlap*. If an overlap lies entirely to the right of the y-axis, we call it a *right overlap*.

Observe that if a left overlap is part of an optimal solution OPT then there is no right segment of OPT that intersects with the region to the left of the overlap. This region thus induces a tripartite instance of MIS-ORTH-OSEG. Similarly, the region to the right of a right overlap of OPT induces a tripartite instance of MIS-ORTH-OSEG. In the following, we show that the region between the two overlaps can be decomposed into two tripartite instances of MIS-ORTH-OSEG. For this, we will consider special (left and right) overlaps.

Lemma 7.4. *Let $\mathcal{I}$ be an instance of* MIS-ORTH-OSEG *and let OPT be an optimal solution for it. If OPT contains a left overlap, then $\mathcal{D}$*

can be partitioned into regions R_1, R_2, and R_3 such that

- $\mathcal{I}[R_1]$ *is a tripartite instance of* MIS-ORTH-OSEG
- $\mathit{OPT}[R_2]$ *does not contain a left overlap*
- $\exists f, c \in \mathit{OPT}$ *s.t.* $\mathit{OPT} = \mathit{OPT}[R_1] \cup \mathit{OPT}[R_2] \cup \{f, c\}$

Proof. Assume that OPT contains a left overlap. Each overlap consists of two segments: the one further from the y-axis, which we call the *far* segment, and the one closer to the y-axis, which we call the *close* segment. Let $\{f, c\}$ be the left overlap in OPT where the far segment f is the rightmost far segment occurring in a left overlap of OPT, and where further the close segment c is the segment that is closest to f among all $c' \in$ OPT that form a left overlap $\{f, c'\}$.

Let E be the rectangle from the free-endpoint of c to the free-endpoint of f. Due to the choice of f and c, no segments in OPT lie within E. Let R_1 and R_2 be the region to the left and right of $f \cup c \cup E$, respectively, such that R_1, R_2, and $f \cup c \cup E$ are a partition of $\mathcal{D}$. See Figure 7.3 for illustration.

Clearly, $\text{OPT} = \text{OPT}[R_1] \cup \text{OPT}[R_2] \cup \{f, c\}$. Since $\{f, c\}$ is a left overlap, $\mathcal{I}[R_1]$ does not contain a right segment, and thus it is a tripartite instance of MIS-ORTH-OSEG.

Now let g be the chord of $\mathcal{D}$ containing f. Note that by the choice of f, for each left overlap in OPT, its far segment lies to the left of or on g. Thus, no pair of segments of $\text{OPT}[R_2]$ can form a left overlap. This completes the proof. □

Theorem 7.5. *Let $\mathcal{I}$ be an instance of* MIS-ORTH-OSEG *and let OPT be an optimal solution for it. There is a set of segments $S \subseteq \mathit{OPT}$ that allows to determine in polynomial time pairwise disjoint instances $\mathcal{I}_1, \ldots, \mathcal{I}_h \subseteq \mathcal{I}$ of tripartite* MIS-ORTH-OSEG*, such that*

$$|\mathit{OPT}| = |S \cup \bigcup_{i=1}^{h} \mathit{OPT}(\mathcal{I}_i)|$$

Moreover, $|S|$ is bounded from above by a constant.

Proof. The set S that we will construct in the following separates $\mathcal{D}$ into a constant number of regions. The i'th region determines $\mathcal{I}_i$ as a

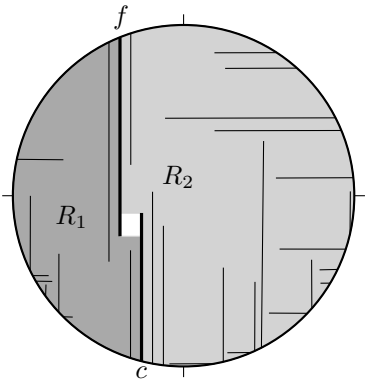


Figure 7.3: Illustration for Lemma 7.4. A left-overlap $\{f, c\}$, yielding a tripartite subproblem $\mathcal{I}[R_1]$. Due to the choice of f and c, no segment can cross the boundary of the white rectangle.

subset of $\mathcal{I}$ contained in that region. We proceed with the construction of S.

Lemma 7.4 shows that we may focus on the case when OPT contains neither a left nor a right overlap: If OPT contains a left overlap then $\mathcal{I}$ can be decomposed into two independent subproblems, namely a tripartite MIS-ORTH-OSEG instance $\mathcal{I}_1$ and an instance $\mathcal{I}'$ of MIS-ORTH-OSEG, each induced by segments lying completely inside the region to the left and, respectively, right of the overlap. Thus, it suffices to consider $\mathcal{I}'$. Lemma 7.4 also states that $\mathcal{I}'$ admits an optimum not containing a left overlap. Symmetrically, we can use Lemma 7.4 to eliminate right overlaps in the optimal solution. Hence, we further consider only the case where OPT contains neither a left nor a right overlap.

We distinguish two cases: first, we consider the case when OPT contains a vertical overlap, i.e., consisting of both a western and eastern vertical segment, and then the case when OPT does not contain a vertical overlap.

Case 1. We assume that OPT contains a vertical overlap. Let t be a top segment in OPT with the lowest y coordinate of its free-endpoint. Let b be a bottom segment in OPT with the highest y coordinate of its

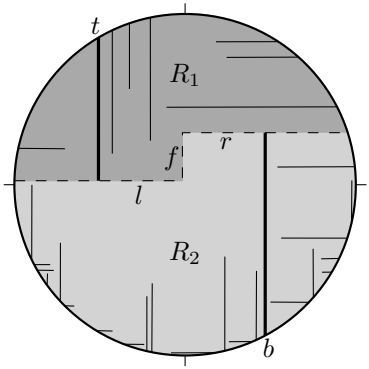


Figure 7.4: Ilustration to Theorem 7.5. Segments t and b yield a separation line.

free-endpoint. Due to our assumption that OPT contains an overlap, t and b overlap in particular. Since there is no left and no right overlap, t and b lie on different sides of the y-axis. We assume w.l.o.g. that t lies to the left of the y-axis and b lies to the right of the y-axis. Let l be the horizontal line connecting the y-axis with the boundary of $\mathcal{D}$ passing through the free-endpoint of t and let r be the horizontal line connecting the y-axis with the boundary of $\mathcal{D}$ passing through the free-endpoint of b (see Figure 7.4). Let e be the line on y axis connecting the endpoints of l and r.

Observe that there are no top segments in OPT below l or crossing l, because a top segment in OPT below or crossing l would have the y-coordinate of its free-endpoint lower than the y-coordinate of the free-endpoint of t, a contradiction to the choice of t. Also, there is no top segment crossing or below r: any such segment would form a right overlap with r, a contradiction to our assumption. Similarly, there are no bottom segments in OPT above or crossing l or r. Since l and r are horizontal, only vertical segments could possibly cross them. Therefore no segments in OPT cross l or r.

Now observe, that no (horizontal) segment in OPT crosses e, as it would have to cross either t or b. The curve consisting of l, e and r divides $\mathcal{D}$ into two regions R_1 and R_2 that lie above and below the curve, respectively. These regions separate OPT into two independent

parts. The part of OPT contained in R_1 is an optimal solution for instance $\mathcal{I}_1 = \mathcal{I}[R_1] \setminus \mathcal{B}$. The part of OPT contained in R_2 is an optimal solution for $\mathcal{I}_2 = \mathcal{I}[R_2] \setminus \mathcal{T}$. Both $\mathcal{I}_1$ and $\mathcal{I}_2$ are tripartite instances of MIS-ORTH-OSEG. This completes the proof of the theorem for Case 1.

Case 2. We assume that OPT does not contain a vertical overlap. Consider the horizontal line l passing through the free-endpoint of the bottom segment in OPT with the highest y-coordinate of its free-endpoint. Clearly, l separates the top segments in OPT from the bottom segments in OPT. Thus, l divides $\mathcal{D}$ into two regions R_1 and R_2 lying above and below l, respectively. These regions separate OPT into two independent parts. Again, the part of OPT contained in R_1 is an optimal solution for instance $\mathcal{I}_1 = \mathcal{I}[R_1] \setminus \mathcal{B}$. The part of OPT contained in R_2 is an optimal solution for $\mathcal{I}_2$ defined as $\mathcal{I}[R_2] \setminus \mathcal{T}$. Both $\mathcal{I}_1$ and $\mathcal{I}_2$ are tripartite instances of MIS-ORTH-OSEG. This completes the proof of the theorem for Case 2 (and thus of the whole theorem). □

Corollary 7.6. MIS-ORTH-OSEG *can be solved in polynomial time given a polynomial time algorithm for tripartite* MIS-ORTH-OSEG.

Combining the results of the previous sections, we state our main result:

Theorem 7.7. MIS-ORTH-OSEG *can be solved in polynomial time.*

We remark that the algorithms developed in this chapter require a geometric representation of the graph, even if it is known that the graph under consideration is a ORTH-OSEG.

Chapter 8

Crew Swapping, Algorithms and Complexity

8.1 Introduction

Crew scheduling for railways deals with the question what sequence of trips each of the many crews of a railway company should perform each day. This problem has been extensively studied. Different approaches and techniques have been proposed to tackle this complex planning problem with impressive results: high quality solutions for instances of over 15,000 tasks can be found [1, 61, 62]. For this reason, practitioners and researchers have begun to ask more demanding questions: Is it possible to not only find high quality solutions to these problems, but also to model and improve the robustness of such solutions? One crucial aspect for robust crew schedules is how well the schedule can be fixed when trains and thus crews are delayed. The goal here is to prevent the follow-up train of the crew from being delayed. The reason for this objective is that a train might have to wait in the station for its crew if this arrives with a delayed train.

We focus on a specific pragmatic recovery operation for this case, which was suggested by Shebalov and Klabjan [78] for airline crew scheduling: *crew swapping*. The idea of crew swapping is depicted

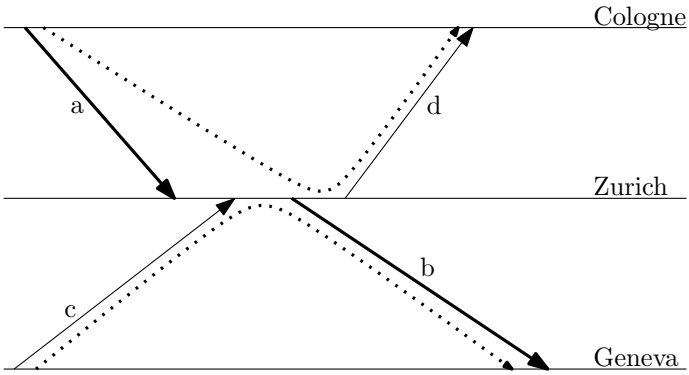


Figure 8.1: Illustration of a crew swap: planned schedule (a-b and c-d, solid) and delayed/recovered schedule (dashed). The crew of a is delayed, so that it cannot start b on time. A possible recovery operation is to use the crew of c as a move-up crew, thus swapping d for b with the other crew.

in Figure 8.1. A crew arrives late at a hub such that it cannot reach its follow-up flight. Instead, it flies a later flight and a second crew, called the *move-up crew*, covers the flight of the delayed crew. Such an exchange is of course only possible under certain conditions, e.g., the two crews need to have the same crew base (to which they return after duty), the maximum labor time may not be exceeded, etc.

This technique is also applicable in a railway setting. The main difference, apart from the different regulatory rules for crew scheduling, is the bigger instance size and the fact that the station network does not decompose into hubs and spokes as easily as an airline network does. While Klabjan and Shebalov propose a large scale integer program for the construction of crew schedules in the airline case, we focus more on the underlying algorithmic questions that arise in the railway case. These questions are related to the following two tasks:

1. Develop a crew schedule that maximizes the number of possible crew swaps.
2. In a situation where a lot of traffic is delayed, decide which crew swaps to perform.

In this chapter, we focus on the second question. We show how to

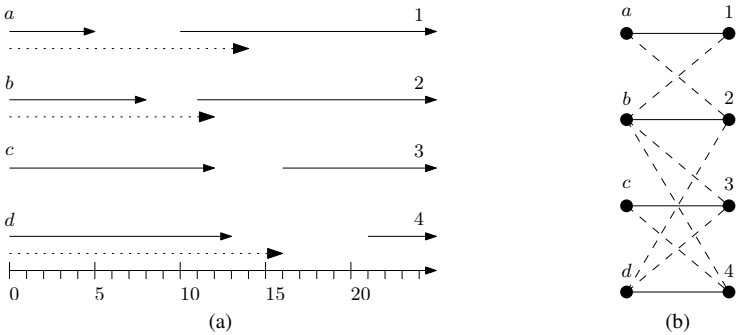


Figure 8.2: Example for a single station. (a) Nominal arrival and departure times of trains (heads and tails of solid arrows) as well as inbound delays (heads of dotted arrows). (b) Transformation into graph G_{IO}. Solid edges represent planned follow up trains P, dashed edges possible move-ups $M \subseteq I \times O$.

formalize it as a *minimum delay propagating crew swapping* (MDCS) problem. We show how to solve the problem for a single station and give complexity results for different variants in a network.

8.1.1 Problem Definition

For a given crew schedule (defining the trips covered by each crew) we are interested in finding optimal crew swap decisions when the schedule is in operation. The objective is to minimize the weighted total delay depending on these decisions. The weights correspond to the importance of the trains, which could typically be determined by the number of expected passengers for each connection. We present a definition of the problem for a single station and for networks of such stations, see Figures 8.2 and 8.3.

Definition 8.1 (Local MDCS)**.** *For a fixed station we are given sets I and O of n inbound and n outbound trains, w.l.o.g., $I = \{1, \ldots, n\}$ and $O = \{1, \ldots, n\}$. The inbound trains have nominal arrival times $\{t_1^{arr}, \ldots, t_n^{arr}\}$ and nonnegative delays $\{\delta_1^{in}, \ldots, \delta_n^{in}\}$. The outbound trains have nominal departure times $\{t_1^{dep}, \ldots, t_n^{dep}\}$ and weights $\{w_1, \ldots, w_n\}$.*

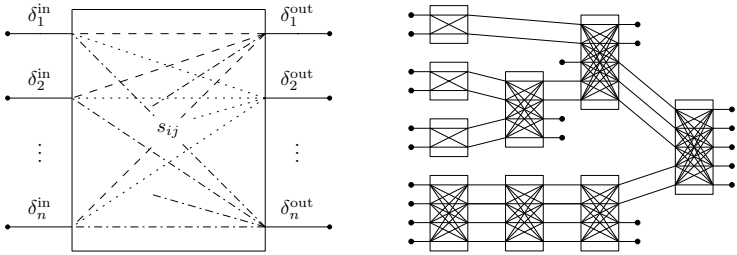


Figure 8.3: A single SE with initial delays δ_i^{in}, outbound delays δ_j^{out}, and slacks s_{ij}. Terminal inputs and terminal outputs are depicted by dots (a). SEs can form a network (b).

Let $P \subset I \times O$ indicate the planned follow-up trains of the respective crew of each inbound train, so that P is a perfect matching in the complete bipartite graph G_{IO} with vertices (I, O). Finally, we are given the set of possible move-ups *$M \subset I \times O$, where move-up (i, j) means that the crew of inbound train i can potentially take the outbound train j. The task is to find a new* crew matching *$P' \subset P \cup M$, P' a perfect matching in G_{IO} that minimizes the weighted sum of delays of the outbound trains. This sum equals*

$$\Delta(P') := \sum_{(i,j)\in P'} w_j \max\left\{0, t_i^{arr} + \delta_i^{in} - t_j^{dep}\right\}. \tag{8.1}$$

In reality, delays might propagate through the network of stations, incurring dependencies among decisions at different stations. To prepare the network version of the above problem, we first introduce a compact representation of the possible swaps at a single station.

We represent a single station of Definition 8.1 by a *shuffle element (SE)* with inputs I and outputs O that are connected via the complete bipartite graph G_{IO}, see Figure 8.3(a). Within each SE, inbound delays are then propagated along the edges. For each edge (i, j), we define the *slack* s_{ij} as $t_j^{\text{dep}} - t_i^{\text{arr}}$ if $(i, j) \in P \cup M$, and as $-\infty$ if the corresponding move-up is impossible. The delay of an outbound train j which is served by the crew of inbound train i is defined as $\delta_j^{\text{out}}(i) := \max\left\{0, \delta_i^{\text{in}} - s_{ij}\right\}$, i.e., an edge $(i, j) \in P'$ will propagate the delay from train i to train j, using up the slack of the edge as much

as possible.

To model the real railway network, outputs and inputs of SEs can be connected by edges, in which case we assume an unchanged propagation of delay, e.g., if an output of SEs α and an input of SE β are connected by an edge (u, v), then $\delta^{\text{out}}_{u,\alpha} = \delta^{\text{in}}_{v,\beta}$. We call outputs of SEs that are not connected to further SEs *terminal outputs* and inputs of SEs that have no predecessor SE *terminal inputs*. By these definitions we can construct an acyclic graph out of SEs, where the direction of an edge corresponds to the direction of travel of the corresponding train between two stations. With the abstraction of a sequence element and the above definitions, it is now easy to state our model for an optimal choice of crew swaps in a snapshot of a railway network, in which a number of trains have specific delays.

Definition 8.2 (Network MDCS)**.** *Given an acyclic graph composed of shuffle elements (SE) and connections between them, delays for all the terminal inputs of this graph and weights for all outputs of SEs, choose a perfect matching for each SE such that the weighted sum of the resulting delays at the outputs is minimum.*

8.1.2 Summary of Results

We address a theoretical abstraction of the problem of making optimal crew swap decisions during operations. We give efficient algorithms for the local case and show that optimizing crew swaps over the whole railway network is NP-hard.

8.2 Choosing Optimal Crew Swaps

In the following, we assume an operational scenario in which trains are delayed and a crew schedule allowing for move-up crews is given. We first describe how to locally minimize propagation of delays for a single station. We then give complexity results for the MDCS in networks.

8.2.1 Local MDCS

If only a single station is to be considered, the optimal crew swaps can be computed efficiently. In the weighted case, a perfect weighted

matching in the bipartite graph corresponding to the station, as described in Definition 8.1, can be computed in $O(\sqrt{n}m \log(nW))$ [68]. In the unweighted case, we can do even better:

Lemma 8.3. *An optimal crew swap at a single station with unit weights can be computed in $O(n \log n)$ by matching trains first-in-first-out (FIFO), according to their actual arrival times $t_i^{\overline{arr}} := t_i^{arr} + \delta_i^{in}$, $i \in I$, and planned departure times t_j^{dep}, $j \in O$.*

Proof. We show that any optimal matching P_{opt} can be transformed into a FIFO matching P_{fifo} without increasing the total outbound delay. We assume w.l.o.g. that trains are sorted according to their actual arrival and planned departure times, i.e., for any $i, j \in I$ we have $i < j \Rightarrow t_i^{\overline{\text{arr}}} \leq t_j^{\overline{\text{arr}}}$, and for any $i, j \in O$ we have $i < j \Rightarrow t_i^{\text{dep}} \leq t_j^{\text{dep}}$. Hence, $i = j$ for all $(i, j) \in P_{\text{fifo}}$.

Given an optimal matching P_{opt} that is not a FIFO matching, let a be the first inbound train that is not FIFO matched to outbound train c, $a = c$, but instead is matched to an outbound train d with $c < d$. Hence, there are matching edges $(a, d), (b, c) \in P_{\text{opt}}$, matching outbound train c to some inbound train b with $a < b$. By swapping the matching of inbound trains a, b to outbound trains c, d, we get the matching $P'_{\text{opt}} := (P_{\text{opt}} \setminus \{(a, d), (b, c)\}) \cup \{(a, c), (b, d)\}$. In P'_{opt}, the first non-FIFO matched inbound train i, if any, must arrive later than a, so $a < i$. By repeatedly swapping matching edges in this way, all trains will be FIFO-matched eventually. Hence, if we can show that the total delay $\Delta(P_{\text{opt}}) \geq \Delta(P'_{\text{opt}})$, it follows that $\Delta(P_{\text{opt}}) \geq \Delta(P_{\text{fifo}})$.

Since we only exchange two pairs of edges, $\Delta(P_{\text{opt}}) \geq \Delta(P'_{\text{opt}})$ is equivalent to

$$(a - d)^+ + (b - c)^+ \geq (a - c)^+ + (b - d)^+ \tag{8.2}$$

where $(u - v)^+ := \delta_v^{\text{out}} = \max\left\{0, t_u^{\overline{\text{arr}}} - t_v^{\text{dep}}\right\}$ denotes the outbound delay of train v propagated via matching edge (u, v). If $t_a^{\overline{\text{arr}}} > t_d^{\text{dep}}$ or $t_b^{\overline{\text{arr}}} < t_c^{\text{dep}}$, Inequality 8.2 holds with equality. Otherwise, $t_a^{\overline{\text{arr}}} \leq t_d^{\text{dep}}$ and $t_b^{\overline{\text{arr}}} \geq t_c^{\text{dep}}$, so the LHS of Inequality 8.2 equals $L := t_b^{\overline{\text{arr}}} - t_c^{\text{dep}}$. Then on the RHS of Inequality 8.2, either both terms equal zero, or only $(a - c)^+$ is positive, implying $L \geq t_a^{\overline{\text{arr}}} - t_c^{\text{dep}}$ since $a < b$, or only $(b - d)^+$ is positive, implying $L \geq t_b^{\overline{\text{arr}}} - t_d^{\text{dep}}$ since $c < d$, or

both terms are positive, implying $L \geq t_a^{\overline{\text{arr}}} - t_c^{\text{dep}} + t_b^{\overline{\text{arr}}} - t_d^{\text{dep}}$ since $t_a^{\overline{\text{arr}}} \leq t_d^{\text{dep}}$. □

8.2.2 Network MDCS

Solving the MDCS for a network of SEs is, in general, NP-hard. In the following, we show that it is even NP-hard in the unweighted case of only two SEs with an arbitrary number of inputs, as well as in the case of an arbitrary number of SEs with only two inputs and outputs. Together, these two results indicate that the complexity of the network version lies both in the size of the single SEs and in the dependencies of local crew swap decisions between stations even in the most simplistic network topologies.

We start with a "technical" lemma that shows what kind of slack values for a SE can arise from a given set of arrival and departure times.

Lemma 8.4. *A set of finite slack values s_{ij} between inputs $i \in I$ and outputs $j \in O$ can be implemented by a single station instance, i.e., by a tuple $\Big(\{t_i^{arr} : i \in I\}, \{t_j^{dep} : j \in O\}, \{w_j : j \in O\}\Big)$, if and only if there exists a constant c such that for all perfect matchings $P \in \mathcal{P}$ of G_{IO} it holds that $\sum_{(i,j)\in P} s_{ij} = c$.*

Proof. Given a SE, the sum of all slacks of any perfect matching $P \in \mathcal{P}$ is constant, since

$$\sum_{(i,j)\in P} s_{ij} = \sum_{(i,j)\in P} t_j^{\text{dep}} - t_i^{\text{arr}} = \sum_{j\in O} t_j^{\text{dep}} - \sum_{i\in I} t_i^{\text{arr}} \ .$$

Given a set of finite slack values s_{ij} for $i \in I$ and $j \in O$, we set $t_1^{\text{arr}} = 0$ and then $t_1^{\text{dep}} := t_1^{\text{arr}} + s_{1,1}$, $t_j^{\text{dep}} := t_1^{\text{arr}} + s_{1j}$ for all $j \in O \setminus \{1\}$, and finally $t_i^{\text{arr}} := t_1^{\text{dep}} - s_{i1}$ for all $i \in I \setminus \{1\}$. We need to show that every other slack value s_{ij} not used in these definitions fits in with the arrival and departure times, i.e., $s_{ij} = t_j^{\text{dep}} - t_i^{\text{arr}}$. To that end, observe that every matching $P \in \mathcal{P}$ can be transformed into any other matching $P' \in \mathcal{P}$ by a series of 2-swaps, i.e., replacing edges $(i,j),(u,v)$ by $(i,v),(u,j)$. Then, $s_{ij} + s_{uv} = s_{iv} + s_{uj}$ must hold since by assumption $\sum_{(i,j)\in P} s_{ij} = \sum_{(i,j)\in P'} s_{ij}$. Especially, for every slack value s_{ij}, $1 < i, j \leq n$, $s_{ij} = s_{i1} + s_{1j} - s_{1,1}$ must hold.

By the definitions above, we have $s_{i1} = t_1^{\text{dep}} - t_i^{\text{arr}}$, $s_{1j} = t_j^{\text{dep}} - t_1^{\text{arr}}$, and $s_{11} = t_1^{\text{dep}} - t_1^{\text{arr}}$, implying $s_{ij} = t_j^{\text{dep}} - t_i^{\text{arr}}$ as demanded. □

After these preliminaries, we turn to the NP-hardness proofs.

Theorem 8.5. *MDCS is NP-hard even in the unweighted case, considering only two consecutive SEs in the network.*

Proof. The proof is by reduction from numerical 3-dimensional matching (N3DM), as defined below, which is known to be NP-complete in the strong sense.

Definition 8.6 (Numerical 3-Dimensional Matching)**.** *[41, SP16] Given disjoint sets X, Y, and Z, each containing n elements, a size $s(a) \in \mathbb{Z}^+$ for each element $a \in X \cup Y \cup Z$, and a bound $B \in \mathbb{Z}^+$, can $X \cup Y \cup Z$ be partitioned into n disjoint sets $A_1, A_2, \ldots, A_n$ such that each A_i, $1 \leq i \leq n$, contains exactly one element from each of X, Y, and Z and such that, for $1 \leq i \leq n$, $\sum_{a \in A_i} s(a) = B$?*

The idea behind the reduction is to construct an instance of MDCS such that an instance of N3DM has a solution if and only if in the corresponding instance of MDCS it is possible that all outbound delays of the last SE are 0. We assume w.l.o.g. that an instance of N3DM satisfies $\sum_{a \in X \cup Y \cup Z} s(a) = nB$.

Let $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, $Z = \{z_1, z_2, \ldots, z_n\}$, s, and B be an instance of N3DM. The corresponding instance of MDCS is defined by two SEs α and β, i.e., I_γ, O_γ, $\delta_{i,\gamma}^{\text{in}}$, $t_{i,\gamma}^{\text{arr}}$, $t_{j,\gamma}^{\text{dep}}$, $w_{j,\gamma}$ for $\gamma \in \{\alpha, \beta\}$, $1 \leq i, j \leq n$, as follows: All inputs of SE α are terminal inputs, all outputs of SE β are terminal outputs, and each output $j \in O_\alpha$ is connected to the corresponding input $j' \in I_\beta$, see Figure 8.4. Each element x_i corresponds to the initial delay $\delta_{i,\alpha}^{\text{in}} := B - s(x_i)$ at SE α. For each element y_j, there are n edges $(i, j) \in I_\alpha \times O_\alpha$ at SE α with slack $s_{ij,\alpha} := s(y_j)$, $i \in \{1, \ldots, n\}$. Similarly, for each element z_k, there are n edges $(j', k) \in I_\beta \times O_\beta$ at SE β with slack $s_{jk,\beta} := s(z_k)$, $j \in \{1, \ldots, n\}$. By Lemma 8.4, this choice of slacks corresponds to an MDCS instance. All weights have unit weight. Next, we show that by the above definitions, N3DM has a solution if and only if the corresponding instance of MDCS has an objective value of $nB - \sum_{a \in X \cup Y} s(a)$.

(1) Assume that a solution of N3DM is given w.l.o.g. by sets $A_i = (x_i, y_i, z_i)$, $1 \leq i \leq n$, with $B = s(x_i) + s(y_i) + s(z_i)$. A solution

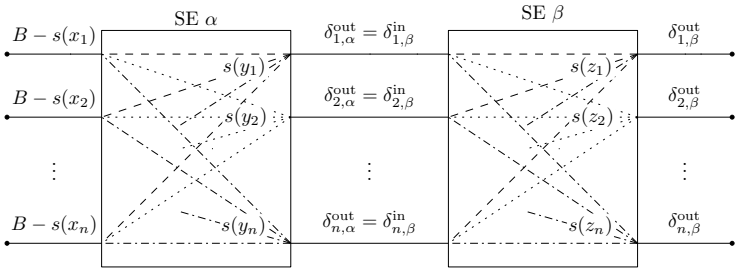


Figure 8.4: Two consecutive SEs in the reduction from N3DM.

of the corresponding instance of MDCS yields for each A_i a disjoint path from a terminal input to a terminal output. For each A_i, $1 \leq i \leq n$, we have $\delta^{\text{in}}_{i,\alpha} = B - s(x_i)$, $\delta^{\text{in}}_{i,\beta} = \delta^{\text{out}}_{i,\alpha} = B - s(x_i) - s(y_i)$, and $\delta^{\text{out}}_{i,\beta} = B - s(x_i) - s(y_i) - s(z_i) = 0$. Hence, $\sum_{\gamma \in \{\alpha,\beta\}, 1 \leq i \leq n} \delta^{\text{out}}_{i,\gamma} = nB - \sum_{a \in X \cup Y} s(a)$.

(2) Assume that an optimal solution of the corresponding instance of MDCS with objective value $z^* = nB - \sum_{a \in X \cup Y} s(a)$ is given by the choice of matching edges $(i,j) \in P'_\alpha \subseteq I_\alpha \times O_\alpha$ and $(j',k) \in P'_\beta \subseteq I_\beta \times O_\beta$. The optimal solution yields for each terminal input a path to a terminal output. For any such (i,j,j',k)-path the delays at the outputs of SE α and SE β are $\delta^{\text{out}}_{j,\alpha} = \max\{0, B - s(x_i) - s(y_j)\}$, and $\delta^{\text{out}}_{k,\beta} = \max\{0, B - s(x_i) - s(y_j) - s(z_k)\}$. Denote the objective function by $z = \Delta_\alpha + \Delta_\beta$, with $\Delta_\gamma := \sum_{1 \leq i \leq n} \delta^{\text{out}}_{i,\gamma}$, $\gamma \in \{\alpha, \beta\}$. Hence, for the given optimal solution, we have $\Delta_\alpha = z^* - \Delta_\beta$. By definition of the outbound delays and B, we have $\Delta_\alpha \geq nB - \sum_{a \in X \cup Y} s(a) = z^*$, and $\Delta_\beta \geq nB - \sum_{a \in X \cup Y \cup Z} s(a) \geq 0$.

Suppose $\Delta_\beta > 0$, then $\Delta_\alpha < z^*$, contradicting $\Delta_\alpha \geq z^*$. Therefore, $\Delta_\beta = 0$, meaning that on each disjoint (i,j,j',k)-path defined by P', the initial delay is consumed by the slacks along that path, i.e., $B \leq s(x_i) + s(y_j) + s(z_k)$. This must hold with equality, since otherwise summing up over all paths would yield $nB < \sum_{a \in X \cup Y \cup Z} s(a)$, a contradiction to the property of the instance of N3DM that $\sum_{a \in X \cup Y \cup Z} s(a) = nB$. □

Theorem 8.7. *MDCS is NP-hard even in the case of unweighted consecutive SEs with only two inputs and outputs.*

Proof. The proof is by reduction from Partition to MDCS.

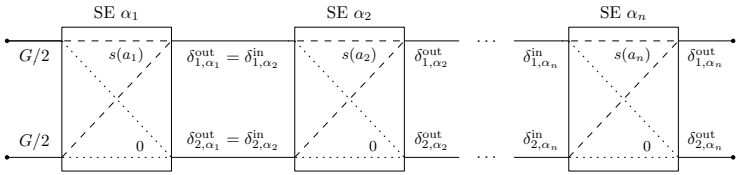


Figure 8.5: Consecutive SEs with only two inputs and outputs each.

Definition 8.8 (Partition)**.** *[41, SP12]*
Given a finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$?

The idea behind the reduction is that an instance of Partition has a solution if and only if in the corresponding instance of MDCS it is possible that all outbound delays at the last SE are 0.

Let $A = \{a_1, a_2, \ldots, a_n\}$, $s(a)$ be an instance of Partition. The corresponding instance of MDCS is defined by n SEs α_ℓ, I_{α_ℓ}, O_{α_ℓ}, $\delta^{\text{in}}_{i,\alpha_\ell}$, $t^{\text{arr}}_{i,\alpha_\ell}$, $t^{\text{dep}}_{j,\alpha_\ell}$, w_{j,α_ℓ}, $\ell \in \{1, \ldots, n\}$, as follows: All inputs of SE α_1 are terminal inputs, all outputs of α_n are terminal outputs, and each output $j \in O_{\alpha_\ell}$ is connected to input $j \in I_{\alpha_{\ell+1}}$ for $1 \leq \ell \leq n-1$, see Figure 8.5.

Each element $a_i \in A$ corresponds to the slacks of SE α_ℓ by setting $s_{j1,\alpha_\ell} := s(a_i)$, $s_{j2,\alpha_\ell} := 0$, $j \in \{1, 2\}$. The initial delays are set to $\delta^{\text{in}}_{j,\alpha_1} := G/2$, $j \in \{1, 2\}$, with $G := \sum_{a \in A} s(a)$. By Lemma 8.4, this choice of slacks corresponds to an MDCS instance. All weights have unit weight. Define for each SE α_ℓ the total outbound delay $\Delta_{\alpha_\ell} := \delta^{\text{out}}_{1,\alpha_\ell} + \delta^{\text{out}}_{2,\alpha_\ell}$. The objective function of MDCS is $z := \sum_{i=1}^{n} \Delta_{\alpha_\ell}$.

Next, we show that by the above definitions, Partition has a solution if and only if the corresponding instance of MDCS has an objective value of $z^* := \sum_{i=1}^{n-1} \left(G - \sum_{j=1}^{i} s(a_i) \right)$.

(1) Assume that a solution of Partition is given by a set $A' \subseteq A$. A solution of the corresponding instance of MDCS yields two disjoint paths from a terminal input to a terminal output, one for A' and one for $A \setminus A'$. Along the path for A', the initial delay of $G/2$ is successively reduced at each SE α_ℓ by $s(a_i)$ if $a_i \in A'$, and by 0 otherwise. Symmetric arguments hold for the path for $A \setminus A'$. It is easy to see that for any solution of Partition, the total initial delay is

reduced at each SE α_ℓ by exactly $s(a_i)$, i.e., $\Delta_{\alpha_\ell} \geq G - \sum_{j=1}^{i} s(a_j)$ holds with equality for $1 \leq i \leq n$. Hence, $\sum_{i=1}^{n} \Delta_{\alpha_\ell} = z^*$ as desired.

(2) Assume that an optimal solution of the corresponding instance of MDCS with objective value z^* is given by the choice of matching edges P' at each SE. The optimal solution yields two disjoint paths from each terminal input to a terminal output. Along each path, the ingoing delay at each SE α_ℓ is either fully propagated to the output or reduced by at most $s(a_i)$. Rewrite the objective function as $z = \sum_{i=1}^{n-1} \Delta_{\alpha_\ell} + \Delta_{\alpha_n}$. Assume that $\Delta_{\alpha_n} > 0$, then $\sum_{i=1}^{n-1} \Delta_{\alpha_\ell} < z^*$, a contradiction because $\sum_{i=1}^{n-1} \Delta_{\alpha_\ell} \geq \sum_{i=1}^{n-1} \left(G - \sum_{j=1}^{i} s(a_i)\right) = z^*$. Therefore, $\Delta_{\alpha_n} = 0$, meaning that on both disjoint paths defined by P', the initial delay is consumed by the slacks along each path, i.e. $G/2 \leq \sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$. But this must hold with equality, since summing up over both paths, we have $G \leq \sum_{a \in A' \cup A \setminus A'}$, which holds with equality by definition of G. □

8.3 Conclusion

The advantage of the move-up crew concept is that it is one of the easiest recovery operations in practice. For the problems arising during operations, we showed how to optimally decide the crew swaps at a single station. In the general case of a station network, however, it is NP-hard to decide optimally. It would be interesting to find approximation algorithms for this case or to study the problem in an online setting. Further, it is interesting to study how to create robust crew schedules that yield many possibilities to swap crews. This is further investigated in [26].

Summary of Contributions

In this thesis we study several problems related to the planning and optimization of railway operations. In some cases, like the allocation of tracks at hump yards, we seek to model all relevant aspects of a real-world problem. In other cases, as the considered special vertex disjoint paths problems, we are interested in the complexity of abstractions of the underlying problem. Thus, our results range from experimental computations on real-world data to purely theoretical results concerning the complexity and approximability of problems.

This thesis is comprised of several papers which are the work of many authors. In the following, I detail my contribution on each of the chapters and respective papers.

Scheduling Additional Trains on Corridors We develop a combination of linear regression models and a combinatorial shortest path model in order to compute a train path that minimizes the risk of delay of the additional train. We discuss the consequences of different model choices and notions of risk with respect to the algorithmic complexity of the resulting combinatorial problems. In particular, it is NP-complete to compute a train path which minimizes the median risk of delay. We can, however, compute a set of Pareto optimal train schedules with respect to risk of delay and travel time, if we are interested in minimizing the average delay.

This chapter is joint work with Thomas Graffagnino and Marc Nunkesser. The idea to make use of existing delay data to plan an additional train on a corridor is by Thomas Graffagnino from the

Swiss Federal Railways. The statistical and combinatorial models and the complexity results are joint work with Marc Nunkesser. Marc Nunkesser implemented the statistical models in R. The implementation in Java, comprising of the processing of the raw data, the combinatorial algorithms as well as the experiments, were carried out by me. This chapter was published in [28].

Mining Railway Delay Dependencies We present efficient algorithms to detect dependencies between delays of trains due to shared track segments as well as scheduled connections. In particular, by applying our algorithms to real-world delay data, one finds trains that are involved in several such dependencies, suggesting that most of their delay is secondary delay due to other trains.

This chapter is joint work with Rati Gelashvili, Thomas Graffagnino and Marc Nunkesser. Thomas Graffagnino suggested the problem and provided us with data. Rati Gelashvili suggested the basic algorithmic approach for the case of single dependencies. This chapter was published in [27].

Track Allocation in Hump Yards To the best of our knowledge, we are the first to model this particular problem at European hump yards. We give a complexity analysis of the underlying problem variants, and link them to the so called μ-coloring problem in interval graphs. We model the problem as a binary integer program and develop both a construction and an improvement heuristic. In our experiments with real-world data from the Hallsberg hump yard in Sweden, we obtain feasible solutions from the integer program in all scenarios, and from the heuristics in most scenarios.

This chapter is joint work with Markus Bohlin, Jens Maue, and Matúš Mihalák. The results of this chapter have been published in [11]. A preliminary version has been published in [10].

Markus Bohlin approached us with the problem for which he obtained the data. By that time, he had already created a first mixed integer programming formulation for the mixing problem. The preliminary version [10] contains the latter model by Markus Bohlin. Further, Heuristic B in [10] is the work of Jens Maue. The experiments in AMPL were carried out by Markus Bohlin.

In [11], we chose to solve the roll-in problem by a new heuristic

and the mixing problem by a new MIP model, both of which I designed and implemented. The theoretical and experimental results in this chapter are joint work with Markus Bohlin and Matúš Mihalák.

Hardness of MFAS in Euler-Paths This result is joint work with Michael Gatto during a conference in Patras. At the same time, an equivalent proof has been found by Anita Schöbel and Peter Widmayer at the airport of Athens! This proof has not yet been published.

Selecting Vertex Disjoint Paths in Planar Graphs We introduce a variant of the vertex disjoint paths problem in planar graphs where paths have to be selected from given sets of paths. We investigate the problem as a decision, maximization, and routing-in-rounds problem. We prove that all considered variants are NP-hard in planar graphs. We study how restrictions on the location of the terminals on the outer face of the planar embedding of the graph lead to polynomially solvable cases for the decision and maximization versions of the problem. For the routing-in-rounds problem we obtain a p-approximation algorithm if there is a separating cut. This chapter is joint work with Matúš Mihalák, Anita Schöbel, Peter Widmayer, and Anna Zych, and has been published in [29]. Therein, the original version of Theorem 6.6 is the sole work of Anita Schöbel. Theorem 6.2 is the sole work of Anna Zych.

MIS in Outersegment Graphs We present a polynomial-time algorithm for the problem of computing a maximum independent set in outersegment graphs where every segment is either horizontally or vertically aligned, given a geometric representation of the graph as input. This chapter is joint work with Matúš Mihalák, Peter Widmayer, and Anna Zych and has been published in [30].

Move-up Crews The results of this chapter are joint work with Marc Nunkesser and were published in [26]. These theoretical considerations also fit into the context of rolling stock circulation, and are included in a technical report on the integration of rolling-stock circulation and delay management [31], which is joint work with Marc Nunkesser, Michael Schachtebeck, and Anita Schöbel. Some of this work is also presented in the thesis of Michael Schachtebeck [73].

Nomenclature

Part I — Utilizing Delay Data

a_i	planned arrival time of a train at station S_i
b	buffer time
b'	largest delay for which connection is held
$\mathcal{C}$	a collection of subsets
C_i	a subset
d_i	planned departure time of a train from station S_i
$\underline{d}$	earliest departure time
$\overline{d}$	latest departure time
d	day
D	set of days
$e_{i,i+1}$	edge in G_r
$F(v_i)$	Pareto frontier associated with a node
G_r	layered, time expanded graph
$l(a_i, d)$	average net change in delay
$\mathcal{M}_{i-1,i}$	linear regression model, between stations
$p(e_{i,i+1})$	multiplicative constant in edge weight
p_i	planned pass-through time of a train at station S_i

$q(e_{i,i+1})$	additive constant in edge weight
$\overline{r}$	maximal number of allowed exceptional points
r	request for a train
S_i	a station or operating point along a corridor
$s(\tau, d_{i-1}, a_i)$	planned slack time of a train
e	delay
S	set of points
S_E	residual error
U	a ground set
v_i	node in G_r
$\bar{w}$	maximal waiting time for connection
$w_q(a_i, d)$	number of train arrivals and departures
x_d	delay of source train on day d
y_d	delay of victim train on day d
$\hat{\delta}_i(\tau, d, \pi)$	predicted delay
$\delta_{i-1}(\tau, d, \pi)$	delay of train at previous station
$\delta_j^{\text{prev}}(a_i, d)$	delay of neighboring train
$\Delta_j^{\text{prev}}(a_i, d)$	planned difference in time to neighboring train
π	path of a train along a corridor
τ_x	train, source of secondary delay
τ	a train
$\vartheta(\tau)$	type of a train
τ_y	train, receiver of secondary delay

Part II — Optimizing Operations at Classification Yards

$\mathcal{A}_p$	set of relevant pairs of trains and start times for pull-out p
arr_i	minimum roll-in time among all cars of train i
dep_i	time of departure of train i from the classification bowl
c_{is}	number of extra roll-ins of a cars of train i when starting at time s
c	a color
$\mathcal{G}_i$	set of car groups that form train i
$\mathcal{X}_{is}$	set of groups of cars of outbound train i arriving before time s
I_i	interval in time during which cars of an outbound train i are stored in the classification bowl
IJ	set of pairs of trains that result in a cut-off if allocated to the same track
k	number of formation tracks
ℓ_g	physical length of group g
ℓ^{mix}	total length of the mixed tracks
ℓ_κ	length of track κ
L_i	set of feasible tracks for train i
n_g	number of cars in group g.
n^{in}	number of inbound trains
n	number of outbound trains
o_i	End time of the allocation of a classification track for train i.
p_s^+	first pull-out scheduled after time s
$\mathcal{P}$	set of pull-outs

$\mathcal{P}_i$	set of relevant pull-outs for train i
s_i	Start time of the allocation of a classification track for train i.
$\mathcal{S}_i$	set of relevant start times for train i.
v	excess of capacity on the mixed tracks
x_{is}	indicates whether the formation of train i starts at time s
$y_{i\kappa}$	indicates whether train i is allocated to classification track κ
κ	classification track

Part III — Theoretical Models for Dispatching

A	antichain
A_i	region above s_i
$\mathcal{B}$	set of bottom-segments
B_i	region to the side of s_i not containing the lowest point of the disk
$c(P_i)$	color of a path P_i
$\mathcal{D}$	disk
$\mathcal{D}_1, \ldots, \mathcal{D}_r$	partition of $\mathcal{D}$
d	number of directions
$\mathcal{I}$	set of straight line segments
I	set of inbound trains
k	number of terminal pairs
$\mathcal{L}$	set of left-segments
MIS	maximum independent set
O	set of outbound trains
OPT	optimal solution
$\mathcal{P}$	collection of sets of paths
$\mathcal{P}_i$	set of alternative s_i-t_i paths
p	maximum number of alternative paths per terminal pair
P_i	s_i-t_i path
$<_{\mathcal{R}}$	binary relation on paths of $\mathcal{R}$
$\mathcal{R}$	set of right-segments
$\mathcal{R}$	the union of all alternative paths

r	number of rounds
$\{s_i, t_i\}$	terminal pair
s	segment
$S[X]$	segments of S contained in region $X \subset \mathcal{D}$
s_i	segment
$\mathcal{T}$	set of top-segments
T	set of terminal pairs
U_i	region to the side of s_i containing the lowest point of the disk
x_{ij}	indicator variable for choice of path P_j for terminal pair $\{s_i, t_i\}$

Glossary

We briefly define some railway specific terms used in this thesis.

classification yard Station of a rail-freight network with the purpose of formation of outbound trains from cars of inbound trains. Consists of the following sets of tracks: an arrival yard, a classification bowl, and a departure yard.

classification track A track of the classification bowl of a classification yard.

connecting train Train of a connection that passengers change to after getting off the feeder train.

corridor A set of track segments that directly connect two major stations in a railway network.

coupling Activity of physically connecting cars such that they can be transported as a whole by an engine.

decoupling Activity of physically disconnecting cars such that they can be left at a station or that new trains can be formed.

delay Difference in time between a timetabled and corresponding actual event at a station.

dispatching Decision making in the case that delays in order to execute operations as closely as possible to planned timetable. May involve wait or no-wait decision in case of timetabled connections, rerouting and rescheduling of trains, reassignment of crews and rolling stock, or even the cancellation of trains.

feeder train The train of a connection that brings passengers to the station at which they change to the connecting train.

headway A minimum distance that has to be kept between trains for reasons of security.

hump yard Special kind of classification yard in which cars are pushed over a hump such that each car rolls into its designated classification track by means of gravity.

primary delay Delay that is not caused by external events such as customer behavior, weather, technical breakdown, etc., except for secondary delay.

rolling stock Umbrella term for railway engines and cars (wagons).

secondary delay Delay that is caused due to the delay of another train, e.g., due to a timetabled connection or shared track infrastructure.

single wagon load traffic Service offered by rail-freight operators in which customers can send smaller shipments, typically in a hub-and-spoke network. As a consequence, trains may consist of cars of various customers and destinations. Usually, intermediate stops at classification yards are necessary to route each car to its final destination.

station Usually a passenger railway station. Sometimes used to refer to points of interest in the network for which data is available (also called “operating point”).

train Denotes both a composition of rolling stock as well as a timetabled service, allowing the transportation of passengers or goods between stations.

train path An allocation of tracks for a timetabled train in space and time.

Bibliography

[1] Erwin Abbink, Matteo Fischetti, Leo Kroon, Gerrit Timmer, and Michiel Vromans. Reinventing crew scheduling at Netherlands Railways. *Interfaces*, 35(5):393–401, 2005.

[2] Alok Aggarwal, Amotz Bar-Noy, Don Coppersmith, Rajiv Ramaswami, Baruch Schieber, and Madhu Sudan. Efficient routing and scheduling algorithms for optical networks. In *Proceedings of the fifth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 412–423, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors. *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, volume 5868 of *Lecture Notes in Computer Science*. Springer, 2009.

[5] Hirotugu Akaike. A new look at statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[6] Carola Alzén. *Trafikeringsplan Hallsbergs rangerbangård*. Banverket, May 2006.

[7] Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

[8] Annabell Berger, Andreas Gebhardt, Matthias Müller-Hannemann, and Martin Ostrowski. Stochastic delay prediction in large train networks. In Alberto Caprara and Spyros Kontogiannis, editors, *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–111, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[9] Ulrich Blasum, Michael R. Bussieck, Winfried Hochstättler, Christoph Moll, Hans-Helmut Scheel, and Thomas Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, March 1999.

[10] Markus Bohlin, Holger Flier, Jens Maue, and Matúš Mihalák. Hump yard track allocation with temporary car storage. In *The 4th International Seminar on Railway Operations Modelling and Analysis (RailRome)*, 2011.

[11] Markus Bohlin, Holger Flier, Jens Maue, and Matúš Mihalák. Track allocation in freight-train classification with mixed tracks. In Alberto Caprara and Spyros Kontogiannis, editors, *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 20 of *OpenAccess Series in Informatics*, pages 38–51. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

[12] Flavia Bonomo, Guillermo Durán, and Javier Marenco. Exploring the complexity boundary between coloring and list-coloring. *Annals OR*, 169(1):3–16, 2009.

[13] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics Mathematics, 1999.

[14] Carla Conte. *Identifying Dependencies Among Delays*. PhD thesis, Georg-August-Universität zu Göttingen, 2008.

[15] Carla Conte and Anita Schöbel. Identifying dependencies among delays. In I. A. Hansen, A. Radtke, J. Pachl, and E. Wendler, editors, *Proceedings of the 2nd International Seminar on Railway Operations Modeling and Analysis (IAROR)*, 2007.

[16] Sabine Cornelsen and Gabriele Di Stefano. Track assignment. *J. Discrete Algorithms*, 5(2):250–261, 2007.

[17] Winnie Daamen, Rob M. P. Goverde, and Ingo A. Hansen. Non-discriminatory automatic registration of knock-on train delays. *Networks and Spatial Economics*, 9(1):47–69, 2009.

[18] Ido Dagan, Martin Charles Golumbic, and Ron Y. Pinter. Trapezoid graphs and their coloring. *Discrete Applied Mathematics*, 21(1):35–46, 1988.

[19] Elias Dahlhaus, Peter Horák, Mirka Miller, and Joseph F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41–54, 2000.

[20] Elias Dahlhaus, Fredrik Manne, Mirka Miller, and Joe Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proceedings of the Eleventh Australasian Workshop on Combinatorial Algorithms (AWOCA)*, pages 7–16, 2000.

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

[22] Gabriele Di Stefano and Magnus Love Koči. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92:16–33, February 2004.

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

[24] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[25] Stefan Felsner, Rudolf Müller, and Lorenz Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74(1):13–32, 1997.

[26] Holger Flier, Abhishek Gaurav, and Marc Nunkesser. Combinational aspects of move-up crews. In Bernhard Fleischmann, Karl-Heinz Borgwardt, Robert Klein, and Axel Tuma, editors, *Operations Research Proceedings 2008*, pages 569–574. Springer Berlin Heidelberg, 2009.

[27] Holger Flier, Rati Gelashvili, Thomas Graffagnino, and Marc Nunkesser. Mining railway delay dependencies in large-scale real-world delay data. In Ahuja et al. [4], pages 354–368.

[28] Holger Flier, Thomas Graffagnino, and Marc Nunkesser. Scheduling additional trains on dense corridors. In Jan Vahrenhold, editor, *Proceedings of the 8th Symposium on Experimental Algorithms (SEA)*, volume 5526 of *Lecture Notes in Computer Science*, pages 149–160. Springer, 2009.

[29] Holger Flier, Matúš Mihalák, Anita Schöbel, Peter Widmayer, and Anna Zych. Vertex Disjoint Paths for Dispatching in Railways. In *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 14 of *OpenAccess Series in Informatics*, pages 61–73. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[30] Holger Flier, Matúš Mihalák, Peter Widmayer, and Anna Zych. Maximum independent set in 2-direction outersegment graphs. In Petr Kolman and Jan Kratochvíl, editors, *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 6986 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2011.

[31] Holger Flier, Marc Nunkesser, Michael Schachtebeck, and Anita Schöbel. Integrating rolling stock circulation into the delay management problem. Technical Report 132, ARRIVAL Project, `http://arrival.cti.gr/`, 2008.

[32] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[33] David J. Forkenbrock. Comparison of external costs of rail and truck freight transportation. *Transportation Research Part A: Policy and Practice*, 35(4):321–337, 2001.

[34] Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[35] Jacob Fox and János Pach. Coloring K_k-free intersection graphs of geometric objects in the plane. In *Proceedings of the 24th ACM Symposium on Computational Geometry (SoCG)*, pages 346–354, 2008.

[36] Jacob Fox and János Pach. Erdős-Hajnal-type results on intersection patterns of geometric objects. In Ervin Győri, Gyula O. H. Katona, and László Lovász, editors, *Horizons of Combinatorics*, volume 17, pages 79–103. Springer, 2008.

[37] Jacob Fox and János Pach. A separator theorem for string graphs and its applications. *Combinatorics, Probability and Computing*, 19(03):371–390, 2010.

[38] John Fox. *Applied Regression Analysis and Generalized Linear Models*. SAGE, 2nd edition, 2008.

[39] András Frank. Packing paths, cuts and circuits—a survey. In Bernhard Korte, László Lovász, Hans Jürgen Promel, and Alexander Schrijver, editors, *Paths, flows and VLSI-Layout*, pages 49–100. Springer, 1990.

[40] András Frank. Finding minimum weighted generators of a path system. In *Contemporary trends in discrete mathematics: from DIMACS and DIMATIA to the Future*. American Mathematical Society, 1997.

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

[42] Michael R. Garey, David S. Johnson, Gary. L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1(2):216–227, 1980.

[43] Michael Gatto. *On the Impact of Uncertainty on some Optimization Problems: Combinatorial Aspects of Delay Management and Robust Online Scheduling*. PhD thesis, ETH Zürich, No. 17452, 2007.

[44] Michael Gatto, Jens Maue, Matúš Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ahuja et al. [4], pages 310–337.

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

[46] Martin Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[47] Martin Charles Golumbic, Doron Rotem, and Jorge Urrutia. Comparability graphs and intersection graphs. *Discrete Mathematics*, 43(1):37–46, 1983.

[48] Rob M. P. Goverde. Optimal scheduling of connections in railway systems. Technical report, TRAIL, Delft, The Netherlands, 1998.

[49] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics. Springer, 1993.

[50] Shiwei He, Rui Song, and Sohail S. Chaudhry. An integrated dispatching model for rail yards operations. *Computers & OR*, 30(7):939–966, 2003.

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

[52] Mia Hubert, Peter J. Rousseeuw, and Stefan Van Aelst. High-breakdown robust multivariate methods. *Statistical Science*, 23(1):92–119, 2008.

[53] Riko Jacob, Peter Márton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.

[54] Ken-ichi Kawarabayashi and Yusuke Kobayashi. Breaking $O(n^{1/2})$-approximation algorithms for the edge-disjoint paths problem with congestion two. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 81–88. ACM, 2011.

[55] J. Mark Keil. The complexity of domination problems in circle graphs. *Discrete Applied Mathematics*, 42(1):51–63, 1993.

[56] Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010.

[57] Jan Kratochvíl. String graphs. I. The number of critical nonstring graphs is infinite. *Journal of Combinatorial Theory, Series B*, 52(1):53–66, 1991.

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

[59] Jan Kratochvíl and Jiří Matoušek. String graphs requiring exponential representations. *Journal of Combinatorial Theory, Series B*, 53(1):1–4, 1991.

[60] Jan Kratochvíl and Jaroslav Nešetřil. INDEPENDENT SET and CLIQUE problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 31(1):85–93, 1990.

[61] Leo Kroon and Matteo Fischetti. Crew scheduling for netherlands railways: Destination customer. In S. Voss and J.R. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems, pages 181–201. Springer-Verlag, 2001.

[62] Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new Dutch timetable: The OR revolution. *Interfaces*, 39(1):6–17, 2009.

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

[64] Christian Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

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

[66] Matthias Middendorf and Frank Pfeiffer. The max clique problem in classes of string-graphs. *Discrete Mathematics*, 108(1-3):365–372, 1992.

[67] Robert Z. Norman and Michael O. Rabin. An algorithm for a minimum cover of a graph. *Proceedings of the American Mathematical Society*, 10(2):315–319, 1959.

[68] James B. Orlin and Ravindra K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, 54(1-3):41–56, 1992.

[69] János Pach and Géza Tóth. Recognizing string graphs is decidable. *Discrete & Computational Geometry*, 28:593–606, 2002.

[70] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

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

[72] Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[73] Michael Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, Georg-August-Universität Göttingen, 2009.

[74] Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 67(2):365–380, 2003. Special Issue on STOC 2002.

[75] Anita Schöbel. *Optimization in Public Transportation*, volume 3 of *Optimization and Its Applications*. Springer, 2006.

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

[77] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[78] Sergey Shebalov and Diego Klabjan. Roubst airline crew pairing: Move-up crews. *Transportation Science*, 40(3):300–312, 2006.

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

[80] Walter Unger. On the k-colouring of circle-graphs. In *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 294 of *LNCS*, pages 61–72. Springer, 1988.

[81] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[82] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S*. Statistics and Computing Series. Springer, 2003.

[83] Wikipedia. R (programming language) — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=R_(programming_language)&oldid=436649698`, 2011.

[84] Wikipedia. Rail 2000 — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Rail_2000&oldid=419775990`, 2011.

[85] Anja Wille and Peter Bühlmann. Tri-graph: a novel graphical model with application to genetic regulatory networks. Technical report, ETH Zürich, 2004.

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

[87] Jianxin Yuan. *Stochastic modeling of train delays and delay propagation in stations*. PhD thesis, Technische Universiteit Delft, The Netherlands, 2006.