Delay Management in Public Transportation: Capacities, Robustness, and Integration

Dissertation
zur Erlangung des
mathematisch-naturwissenschaftlichen Doktorgrades
„Doctor rerum naturalium“
der Georg-August-Universität Göttingen

vorgelegt von
Michael Schachtebeck
aus Northeim

Göttingen 2009

Referentin:	Prof. Dr. Anita Schöbel
Korreferentin:	Prof. Dr. Sigrid Knust
Tag der mündlichen Prüfung:	17. Dezember 2009

Abstract

In this work, we mainly deal with capacitated delay management that is an important task during the daily operations of a public transportation company. Unlike uncapacitated delay management studied in most publications, it takes into account the limited capacity of the track system in a railway setting and security distances between two trains using the same infrastructure. We introduce a graph-theoretical model for this problem and derive a linear integer program, based on the graph-theoretical model. We prove some important properties of the model which allow us to extend results from the uncapacitated delay management problem to the capacitated case. Furthermore, we use these properties to develop reduction techniques which can significantly reduce the size of an input instance. To be able to solve large-scale real-world instances, we suggest heuristic solution procedures, prove worst-case error bounds, and numerically evaluate all solution approaches within a case study based on real-world data. We show how rolling stock circulations can be integrated in the delay management problem, extend results from capacitated delay management to this integrated problem, prove that it is NP-hard even in very special cases, identify a polynomially solvable case, and suggest a generic solution framework. Apart from delay management, we also consider robustness aspects. We report results from a case study on delay resistant timetabling, resume the concept of recoverable robustness, extend it to multi-stage recoverable robustness, and show how both concepts can be applied to special timetabling problems to compute recoverable-robust timetables. In the end, we present a programming framework for analyzing the impact of different planning stages in public transportation on subsequent planning stages and on the operational phase, for example for analyzing the robustness of a line plan or a timetable in case of delays.

Acknowledgment

First of all, I would like to thank my supervisor Prof. Dr. Anita Schöbel for introducing me to the field of this thesis and for her enduring support. Many fruitful discussions, her helpful advices, and the pleasant working environment contributed considerably to my research. Furthermore, I thank Prof. Dr. Sigrid Knust for acting as the second referee.

I would also like to thank my colleagues in the Institute for Numerical and Applied Mathematics for providing such a stimulating and pleasant environment. I am especially indebted to the members of the Working Group Discrete Optimization (in particular Marc Goerigk, Mark-Christoph Körner, Thorsten Krempasky, Marie Schmidt, and Daniel Scholz) for their help, support, and patience and for sharing many beautiful moments with me, in and outside the office. Special thanks go to Marie Schmidt for proof-reading this thesis and for her helpful comments.

I very much enjoyed the collaboration with my colleagues from all over Europe who participated in the research project ARRIVAL. In particular I thank Serafino Cicerone, Gabriele Di Stefano, Holger Flier, Christian Liebchen, Marc Nunkesser, and Sebastian Stiller for stimulating discussions during our joint research. The publications resulting from our collaboration contributed to this thesis as well. The financial support of the EU Specific Targeted Research Project ARRIVAL is also gratefully acknowledged.

Furthermore, I thank Christian Liebchen and Sebastian Stiller for providing the timetable for the case study and Jens Dupont from Deutsche Bahn AG for providing the underlying data set.

Certainly, I would not be where I am today without the enduring support, encouragement, and patience of my family for which I am deeply grateful.

Above all, my warmest thanks goes to my wife Annika Eickhoff-Schachtebeck: for encouraging me throughout the last years, for believing in me unconditionally, and for sticking by me during times of joy as well as sadness. *Danke.*

Contents

Chapter 1

Introduction

1.1 Overview

Many optimization problems arising from real-world applications consist of two parts: During the *strategic planning phase*, the main goal is a good utilization of available resources, while during the *operational phase*, a good reaction to unforeseen disturbances is required when the system is up and running. The strategic planning phase usually begins long before the system starts to operate, while in the operational phase, a fast (or even real-time) response to unexpected disturbances is required.

In this work, we deal with an example of such a pair of optimization problems: *timetabling* and *delay management* in public transportation. The process of computing a timetable is part of the strategic planning phase. In most publications, the objective during this stage is to minimize the average travel time of passengers as the travel time is a crucial factor in the travelers' choice of mode of transport. Delay management, by contrast, is part of the operational phase: Once the timetable is operated, it is crucial to deal with unforeseen disturbances. As delays significantly reduce the attractiveness of public transportation, the objective of delay management is to minimize the inconvenience of the passengers. In this work, we mainly focus on delay management, i.e. on how to react in case of delays (however, in Chapter 6, we also consider robustness aspects that are important during the timetabling phase).

A delay occurring in the operational phase might have different effects. On the one hand, passengers reaching their final destination with a delayed train end their journey with this delay (if the train is the one they originally planned to take). As long as the delay is not too large, for most passengers, this is no big deal. On the other hand, a delay might also cause some passengers to miss a connection – even if the delay is fairly small (if, for example, time for changing is calculated tightly). In addition, a

single delayed vehicle might affect several other vehicles, for example due to the limited capacity of the track system in a railway setting.

In case of delays, there are different decisions to make:

- If a connection is affected by a delay, i.e. if a feeder train is late, then for each connecting train, one has to decide if the connecting train should wait for the delayed train or if it better departs on time. We call this type of decisions *wait/depart decisions*. In the first case, passengers in the delayed feeder train will catch their connection – however, passengers in the waiting train might get an additional delay (as well as passengers waiting at a subsequent station), and the waiting train might get in conflict with another train using the same piece of track. In the second case, the effects on the rest of the network are more limited; however, passengers missing a connection might suffer a pretty large delay. In practice, wait/depart decisions are often made by applying fixed *waiting time rules* (for example "a local train always waits for a high-speed long-distance train, but at most for ten minutes") as it is done by Deutsche Bahn AG (see [Jac03]) or by pursuing a fixed policy (e.g. a no-wait policy or an all-wait policy). However, fixed policies neither take into account the current situation (for example the number of passengers in a train or the importance of a connection which depends on the number of passengers who actually want to transfer), nor do they consider the impact of the wait/depart decisions on other trains. Thus, incorporating the wait/depart decisions in an optimization scheme can reduce the inconvenience of the passengers.

- In a railway setting, the capacity of the tracks is limited, and trains driving on the same track into the same direction or trains driving into opposite direction on a single track have to respect special security distances. In the original timetable, all those restrictions are taken into account – however, if a train is late, it might get in conflict with another train using the same infrastructure. In this case, one has to decide which train should drive first. This type of decisions is called *priority decisions*. They are necessary to comply with safety regulations and to take into account the limited capacity of the track system. In practice, it is common usage to apply fixed rules for making the priority decisions, for example to prioritize punctual trains as it is done in Sweden (see [Tör08]) or to use fixed priorities for certain types of trains and to prioritize fast trains over slow ones as it is done in Germany (see [Jac08]). Obviously, more sophisticated schemes which take into account the current situation and the effects on other trains are possible and can reduce the inconvenience of the passengers.

The problem of making all wait/depart decisions and all priority decisions and updating the planned timetable to a new *disposition timetable* in such a way that the inconvenience

of the passengers is minimized is called *delay management problem*. As a measure for the inconvenience of the passengers, we use the sum of all delays of all passengers at their final destinations; it can be approximated by a combination of the (properly weighted) number of missed connections and the (properly weighted) sum of all delays of all trains.

Before presenting a model for the delay management problem in Chapter 2, we give an overview of related work.

1.2 Related Work

It is, of course, desirable to reduce the risk of unforeseen disturbances instead of only reacting to delays. To this end, several authors investigate the reasons for primary delays [Zas00, NO00, NH04], the relationship between delays and infrastructure maintenance [Nys05, Nys08], and the frequency distribution of delays and the interaction between different delays [Mat05, Yua06, Güt06, Con08, FGGN09]. However, in practice, it is utterly impossible to totally prevent any primary delay from affecting other vehicles or even to prevent that primary delays occur. Hence, in this work, we focus on how to deal with delays as good as possible during the operational phase and also analyze how to compute the timetable in such a way that it can absorb a given amount of delay, i.e. a timetable that is robust against certain delays.

We start with an overview of different approaches on computing robust timetables.

Robustness Aspects

In [EK04], not only the scheduled waiting time of all passengers, but also their average actual waiting time under random delays is taken into account during the computation of the timetable. As this leads to a cost function that cannot be calculated analytically (but only simulated), the authors use genetic algorithms for solving the resulting optimization problem. In [KDV07], [KMH$^+$08], [KHM08], and [LS09], the authors present stochastic optimization models to investigate how to distribute a limited amount of slack time in a smart way. Different methods for improving the robustness of a given nonperiodic timetable are suggested and compared in [FSZ09]. A local search optimization scheme for improving a train's route within a station to make it more robust against delays is suggested in [CBH05]. In [SK09], robust timetabling is treated as a bicriteria approach. [Her06] analyzes the stability of timetables and the effect of uncertainty on train routing within stations. [CFB$^+$09] deals with how to choose speed profiles of trains in order to optimize the buffer times. Most other models presented in the literature are evaluation

models, i.e. the timetable is modified, the model is applied to evaluate the modification, and the result is used to again modify the timetable. See [KDV07] for an overview of such approaches.

More general robustness concepts that are not limited to applications arising in public transportation are *stochastic programming* (see [BL97, KW94, RS03]) which allows to model optimization problems involving uncertainty and *robust optimization* (see [BS07, BEN06, BS04, FM06]) which aims to find a solution that keeps feasibility when disturbances occur. The concept of *light robustness* that has been introduced in [FM09] adds slacks to the constraints and aims to find a solution that satisfies these relaxed constraints (as robust optimization often leads to solutions that are too conservative and hence too expensive). A first approach to unify the notions of *robustness* and *recoverability* into a new integrated notion of *recoverable robustness* has been suggested in [LLMS09]. In [CDD+07], algorithmic aspects of recoverable robustness have been highlighted by giving the definition of *robust algorithm* and of the corresponding *price of robustness* which is an extension of the price of robustness of a recoverable-robust optimization problem as defined in [LLMS09]. In [CDSS08], recoverable robustness has been extended to *multi-stage recoverable robustness* to take into account several disturbances, occurring one after another. Both concepts have been applied to the timetabling problem in various publications, see [CDSS08, CDD+08, DDNP09, CDD+09a] for the single-stage case, [CDSS08] for multi-stage recoverable-robust timetabling, and [CDD+09b] for an overview containing both models.

All these approaches aim to make the timetable robust against delays. A rather different approach is to focus on how to *react* in case of delays instead of seeking for a robust timetable. In the remainder of this overview, we focus on this aspect.

Delay Management and Re-Scheduling

If all connections and the order of all trains is fixed, i.e. if there are no wait/depart decisions and no priority decisions to make and the only task is to update the timetable to take into account the delays, the problem reduces to a nonperiodic timetabling problem. This problem is easy to solve (see [Roc84] where the corresponding problem is called Feasible Differential Problem), for example by shortest path techniques or linear programming. However, *periodic* timetabling, i.e. the task to compute a timetable where operations repeat in regular intervals (usually each hour or each two hours), is an NP-hard problem, even if the task is only to compute a feasible and not an optimal timetable (see [SU89]). As we do not cover periodic timetabling in this work, we refer to [Lie06] for an overview of different models and solution procedures.

If the order of trains is fixed (or if capacity constraints are neglected), i.e. if no priority decisions have to be made and the remaining task is to make wait/depart decisions and

to update the timetable to a disposition timetable, the problem reduces to the uncapacitated delay management problem. Different non-linear mixed-integer programming (MIP) formulations, approximating the effect of delays on the customers, have been suggested in [Sch01b] and [Kli00]. A first linear integer programming (IP) formulation has been presented in [Sch01a] and has been further developed in [Sch07] and [DHL08], see also [Sch06] for an overview of various models. The complexity of the uncapacitated delay management problem has been investigated in [GGJ+04] and [GJPS05], while the online version of the problem has been studied in [GJPW07], [Gat07], and [BHLS08]. Other publications deal with the application of max-plus algebra for analyzing the propagation of delays and timetable stability, see for example [DDD98], [Gov98], [HdV01], [Gov05], [HOW06] and references therein. In [GS07], delay management is treated as a bicriteria problem; in [HdV01], a bicriteria approach is analyzed by the means of max-plus algebra. Knowledge-based expert systems simulating the effects of wait/depart decisions are treated in [SM97, SM99, SMBG01, SBK01]. A branch and bound approach and kernel-based learning algorithms for solving the uncapacitated delay management problem have been suggested in [Job08]. In [KS09], the efficiency of different dispatching strategies for online delay management has been studied.

If the wait/depart decisions are neglected, the task is to make the priority decisions and to update the timetable to a disposition timetable. In the literature, this problem is called *railway re-scheduling problem*, *train dispatching problem*, or *real-time conflict resolution problem*. It has been studied extensively in the past – however, in many cases, only simple network structures like a single line are considered, and only few publications take into account the passengers' point of view (many publications focus on feasibility or on minimizing delays on a per-train basis only). In [Szp73], one of the earliest publications, the limited capacity of the tracks is taken into account by adding disjunctive ordering constraints to a mathematical program, modeling a train scheduling problem on a long single track line. In [DPP07], the limited capacity of the tracks is modeled by using the alternative graph formulation for job-shop scheduling problems with no-wait and no-store constraints presented in [MP02], based on the concept of disjunctive graphs from [RS64]. In this approach, disjunctive constraints are modeled by a set of alternative arcs. The implementation of a real-time traffic management system using a branch and bound algorithm for computing a good train sequence and local search techniques to improve the solution by re-routing some trains is presented in [DCPP08]. Some approaches do not explicitly take care of track capacities within the model, but enumerate all feasible meeting points of two trains and compare the corresponding solutions to find a good one (see for example [SW83]) or detect and resolve overtaking conflicts by applying fixed rules (as suggested in [CG94]). Other approaches focus on simplified network structures, see for example [HKF96] and [CGM98]. In [BHK02], a rescheduling problem where one track of a double-track line is closed due to construction work is analyzed. A microscopic representation of a railway

network in a station area with multiple conflicting routes and high service frequencies is treated in [CGD09].
The extensive variety of recent solution approaches for the railway re-scheduling problem covers A* search to find a solution that minimizes the weighted delay in all stations [Koc00], genetic algorithms minimizing the total delay [PALF01] or the annoyance for the customers [SW04], greedy approaches minimizing the maximum secondary delay [DP04, MPP04], branch and bound algorithms [DLZL06], heuristics to identify the key modifications needed to minimize the effect of a disturbance [Tör07], and approaches to compute near-optimal solutions by adding additional constraints to an integer programming formulation of the problem [TP07]. In [NYN+05], simulated annealing and program evaluation and review techniques are used to solve a train rescheduling problem, minimizing the dissatisfaction of the customers. For older overviews of different approaches on re-scheduling problems, see [Ass80] and [CTV98], more recent ones are given in [Tör06], [Jac08], and [Lus08].

However, all these studies either focus on pure delay management or on the rescheduling part. Combining both aspects, the problem becomes significantly harder to solve. Some first ideas on how to model the limited capacity of the track system in a railway setting in the context of delay management have been presented in [Sch09b]. Capacity constraints were also taken into account in a real-world application studied within the project DisKon supported by Deutsche Bahn (see [BGJ+05]). Here, the following setting to apply delay management in practice is suggested: In a first step, a macroscopic approach deals with the wait-depart decisions, while a second step ensures feasibility within a microscopic model. However, this approach may yield rather bad solutions. In [SS08], some first ideas for solution procedures have been suggested and further refined in [Sch09a, SS10].

1.3 Outline

The remainder of this work is structured as follows: In Chapter 2, we summarize basic notations and definitions from graph theory which we need later on. Furthermore, we introduce the concept of event-activity networks which is the basic concept for our model. Then we present a linear integer programming formulation of the delay management problem and point out some direct consequences.

After introducing the problem, we analyze the resulting IP formulation in Chapter 3. We derive upper bounds on the "big M" constant which we use to model disjunctive constraints, investigate the headway constraints which make the problem hard to solve, and derive different reduction techniques for reducing the size of an instance of the problem. We extend some results from uncapacitated delay management to the

capacitated case and conclude with numerical results, demonstrating the effectiveness of the suggested reduction techniques.

Although we are able to significantly reduce the input size by the reduction techniques suggested in Chapter 3, it still takes too much time to solve large-scale real-world instances to optimality. Hence, in Chapter 4, we suggest various heuristic solution approaches, prove worst-case error bounds, and present numerical results from a case study, based on real-world data.

Chapter 5 deals with the integration of rolling stock circulations in our model. First, we show how rolling stock circulations can be integrated in the event-activity network and the IP formulation. Then we extend some results from Chapter 3 to the integrated model and prove its hardness even for very special cases. We identify a polynomially solvable case and suggest a generic scheme for solving arbitrary instances.

After focussing on delay management, we deal with robustness aspects in Chapter 6. We summarize some results from a case study on robust timetabling and resume the concept of recoverable robustness. Then we extend this concept to multi-stage recoverable robustness and apply both concepts to a simplified timetabling problem to analyze recoverable-robust timetabling.

We conclude this work with a summary and an outlook to further work and possible extensions in Chapter 7. There, we also present a programming framework which allows to analyze the interaction of different planning stages in public transportation; it is a helpful tool for further research on robust planning.

Chapter 2

The Model

In this chapter, we introduce the model which we use for solving the delay management problem. We start with some basic notations and definitions from graph theory in Section 2.1 and introduce the concept of event-activity networks, the central foundation of our model, in Section 2.2. Finally, in Section 2.3, we describe our model, derive a linear integer programming formulation for it, summarize different relaxations of the IP formulation, and present some theoretical results.

2.1 Basics from Graph Theory

For modeling both the delay management problem and timetabling problems, we use some concepts from graph theory. Throughout this work, we only consider *directed graphs* or *digraphs* $G = (V, E)$ without self-loops, i.e. for each edge $(u, v) \in E$, we require $u \neq v$. In the following, we shortly introduce the most important terms and notations.

Let $G = (V, E)$ be a directed graph. A *directed path* of *length* k from some vertex u to another vertex v in G is a sequence $p = (u = v_1, v_2, \ldots, v_k, v_{k+1} = v)$ of vertices with $(v_i, v_{i+1}) \in E$, $i = 1, \ldots, k$. Sometimes, we write $p \subseteq G$ to emphasize that p is a path in G. A path is called a *directed cycle* if $v_1 = v_{k+1}$. If G does not contain any directed cycle, we call G *acyclic* or *cycle-free*. A directed graph that is acyclic is called a *DAG* (directed acyclic graph). By $\text{pre}(v)$, we denote the set of all *predecessors* of v in G, i.e. the set of all vertices $u \in V$ for which there exists a directed path of length at least 1 from u to v in G. Formally,

$$\text{pre}(v) := \{u \in V \setminus \{v\} \colon \exists\, v_2, \ldots, v_k \in V \colon (u, v_2), (v_2, v_3), \ldots, (v_k, v) \in E\}\,.$$

Analogously, $\mathrm{suc}(v)$ refers to the set of all *successors* of v in G, i.e. the set of all vertices $w \in V$ for which a directed path from v to w in G exists, including v itself. Formally,

$$\mathrm{suc}(v) := \{w \in V\colon \exists\, v_2, \ldots, v_k \in V\colon (v, v_2), (v_2, v_3), \ldots, (v_k, w) \in E\} \cup \{v\}.$$

Note that in the definitions above, v is *not* included in the set $\mathrm{pre}(v)$ of its predecessors, while it *is* included in the set $\mathrm{suc}(v)$ of its successors – this is for technical reasons only to simplify some proofs later on.

Sometimes, we do not want to consider the predecessors or the successors of a node v in the graph G, but in a subgraph $G' = (V, E')$ with $E' \subset E$ (this will get clearer for example in the definition of $\mathcal{E}_{\mathrm{mark}}$ in Section 3.3). In this case, we define the predecessors and the successors of $v \in V$ in G' as

$$\mathrm{pre}(v, E') := \left\{u \in V \setminus \{v\}\colon \exists\, v_2, \ldots, v_k \in V\colon (u, v_2), (v_2, v_3), \ldots, (v_k, v) \in E'\right\}$$
$$\mathrm{suc}(v, E') := \left\{w \in V\colon \exists\, v_2, \ldots, v_k \in V\colon (v, v_2), (v_2, v_3), \ldots, (v_k, w) \in E'\right\} \cup \{v\}.$$

We say a DAG is a *tree* if it is an *out-tree* to some source node $r \in V$ (the *root*), i.e. if there exists a unique path from r to any other node $v \in V$. If $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$, then we call G a *linear graph*.

A *topological ordering* or *topological sort* of a directed graph $G = (V, E)$ is an ordering of its nodes $V = (v_1, v_2, \ldots, v_n)$ such that

$$i < j \;\Rightarrow\; \text{there exists no directed path from } v_j \text{ to } v_i \text{ in } G$$

holds for all vertices $v_i, v_j \in V$. Figuratively speaking, if a path from v_i to v_j exists, then in a topological ordering of G, v_i "appears before" v_j. If a directed graph contains at least one directed circle, then no topological sort of G exists. In contrast, each DAG has at least one topological sort (but it does not have to be unique). For more details, we refer to [CLRS01].

2.2 Event-Activity Networks

To model the delay management problem and to derive solution procedures, we use the concept of *event-activity networks* as suggested in [SU89] (see also [Nac98] for the application of event-activity networks in periodic timetabling and [Sch07] for their application in delay management). An event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is a directed graph whose nodes are called *events* and whose directed edges are called *activities*. Event-activity networks are a widely used mathematical model for periodic or nonperiodic

scheduling of events with time constraints. In the nonperiodic case which we consider here, an activity which connects two events models a precedence constraint between those events: the start event of the activity has to take place first. Each activity has assigned a lower bound on its duration, so the scheduled time of the end event of an activity has to be larger than or equal to the scheduled time of the start event plus the lower bound. In contrast to the nonperiodic case, in periodic event-activity networks (used for example for periodic timetabling), each activity has assigned a lower and an upper bound, modeling time window constraints.

In a railway setting (which is the main focus of our applications), the set $\mathcal{E}$ of events consists of *arrival events* $\mathcal{E}_{\text{arr}}$, i.e. the arrivals of trains at stations, and *departure events* $\mathcal{E}_{\text{dep}}$, i.e. the departures of trains from stations. The set $\mathcal{A}$ of activities consists of four different types of activities:

- *Driving activities* $\mathcal{A}_{\text{drive}} \subset \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{arr}}$ model the driving of a train between two consecutive stations, so a driving activity connects a departure event of a train with its next arrival event at the subsequent station. The lower bound $L_a > 0$ of a driving activity $a \in \mathcal{A}_{\text{drive}}$ represents the minimal driving time between both stations.
- *Waiting activities* $\mathcal{A}_{\text{wait}} \subset \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}}$ represent the waiting of a train within a station, for example for the boarding and deboarding of passengers or for crew change. A waiting activity connects the arrival of a train at a station with its departure from the same station. The lower bound $L_a > 0$ of a waiting activity $a \in \mathcal{A}_{\text{wait}}$ describes the minimal time which is needed to let passengers get on or off and also takes into account the time for crew change or other actions.

Each activity in $\mathcal{A}_{\text{drive}}$ and $\mathcal{A}_{\text{wait}}$ corresponds to an action of one train; since they are all treated in the same way (e.g. the lower bound L_a of each such activity a always has to be respected), we summarize them in the set

$$\mathcal{A}_{\text{train}} := \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}.$$

- *Changing activities* $\mathcal{A}_{\text{change}} \subset \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}}$ allow passengers to transfer from one train to another one within the same station, so a changing activity connects an arrival event of some train at some station with a departure event of another train at the same station. The lower bound $L_a > 0$ refers to the minimum time the passengers need when they transfer between both trains. It is one of the tasks of delay management to decide for each changing activity if the corresponding connection should be maintained or not. If a connection is maintained, the lower bound L_a of the corresponding changing activity a has to be respected, otherwise it can be ignored.

- *Headway activities* $\mathcal{A}_{\text{head}} \subset \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{dep}}$ model the limited capacity of the track system. They always appear in pairs: if $(i, j) \in \mathcal{A}_{\text{head}}$, then $(j, i) \in \mathcal{A}_{\text{head}}$, too. In contrast to the other types of activities, a single headway activity does not model a single constraint, but together with its corresponding counterpart, they model a pair of disjunctive constraints. As an arc in the event-activity network models a precedence constraint, it is not possible to satisfy both constraints resulting from a pair of headway activities at the same time. On the contrary, exactly one headway activity from each pair has to be respected. The goal of delay management hence is to choose exactly one activity of each such pair and to respect the resulting constraint, fixing the order of the two events i and j. If (i, j) is chosen, then event i takes place before event j and the lower bound L_{ij} of activity (i, j) has to be respected. If, however, (j, i) is chosen, then event j takes place first and the lower bound L_{ji} of activity (j, i) has to be respected. The lower bound $L_{ij} > 0$ of a headway activity $(i, j) \in \mathcal{A}_{\text{head}}$ represents a security distance: For the two departure events i and j, it represents the minimal *headway* between the departures of the corresponding trains, i.e. the minimum time for which the train belonging to event j has to wait after the departure of the train belonging to event i to ensure safe operations. Note that those headway times need not to be symmetric; in general, $L_{ij} \neq L_{ji}$ (a slow train probably will block a specific piece of track longer than a fast train). Our model covers two types of limitation: two trains driving on the same track into the same direction and two trains driving into opposite direction on a single-way track.

Summing up, we have

$$\mathcal{E} \;=\; \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$$

and

$$\mathcal{A} \;=\; \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{head}}.$$

To illustrate an event-activity network, we use the following example (which is depicted in Figure 2.1): Assume that we have five stations A, B, C, D, and E. One train drives from station A to station C and further on to station D, while a second train drives from station B to station E via station C. Within station C, passengers might transfer between both trains, and on their way to station C, both trains share a common piece of track.

In the corresponding event-activity network (see Figure 2.2 for an illustration), the arrival of the first train at station C and the departure of the second train from station C are connected by a changing activity; the same holds for the arrival of the second train and the departure of the first train. To model the limited capacity of the tracks, the departure of the first train from station A and the departure of the second train from station B are connected by a pair of disjunctive headway activities.

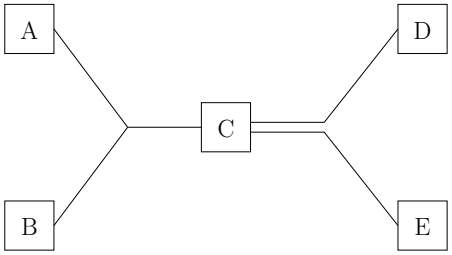

Figure 2.1: An example of five stations A, B, C, D, and E where trains driving between A and C and between B and C share a common piece of track (the solid lines between stations represent tracks).

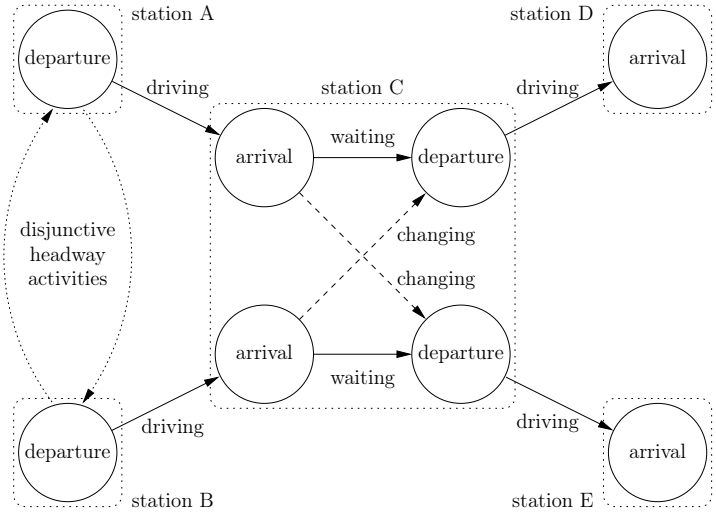

Figure 2.2: The corresponding event-activity network if we assume that one train serves the directed line A-C-D while another one serves B-C-E. Solid arrows are activities from $\mathcal{A}_{\text{train}}$, dashed arrows represent changing activities, dotted arrows represent headway activities.

An equivalent model for headway activities is to connect the arrival event of one train with the departure event of the second train and vice versa, meaning that the second train's departure is not allowed to take place before the first train's arrival. Since in this case, four events instead of two are involved in each pair of headway activities, we do not use this model.

2.3 Model and Integer Programming Formulation

In the following, we use the concept of event-activity networks to give a mathematical formulation of the delay management problem. To this end, we assume that the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ and lower bounds $L : \mathcal{A} \to \mathbb{N}$ are given. A *timetable* is a node potential $\pi : \mathcal{E} \to \mathbb{N}$. It is called *feasible* if it respects the lower bounds of all driving, waiting, and changing activities as well as the lower bound of exactly one headway activity from each pair of disjunctive headway activities, i.e. if π satisfies

$$\pi_j - \pi_i \geq L_a \quad \forall a = (i,j) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \tilde{\mathcal{A}}$$

where $\tilde{A} \subset \mathcal{A}_{\text{head}}$ contains exactly one headway activity from each pair of disjunctive headway activities. Note that the graph $(\mathcal{E}, \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \tilde{\mathcal{A}})$ has to be cycle-free; otherwise, no feasible timetable exists since activities model precedence constraints. Given a feasible timetable π, we call those headway activities that are respected by π, i.e. the headways in $\tilde{\mathcal{A}}$, *forward headways* and those headway activities that are not respected by π *backward headways*. Formally,

$$\begin{aligned} \mathcal{A}^{\text{forw}}_{\text{head}} &:= \{(i,j) \in \mathcal{A}_{\text{head}} : \pi_i < \pi_j\} \\ \mathcal{A}^{\text{back}}_{\text{head}} &:= \{(i,j) \in \mathcal{A}_{\text{head}} : \pi_i > \pi_j\}\,. \end{aligned}$$

The *slack time* s_a of an activity $a \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}^{\text{forw}}_{\text{head}}$ is the time that can be saved if a is performed as fast as possible, i.e.

$$s_a := \pi_j - \pi_i - L_a \geq 0 \quad \forall a = (i,j) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}^{\text{forw}}_{\text{head}}.$$

As in related literature, we distinguish two classes of delays. *Source delays* or *primary delays* are delays caused by external effects like bad weather conditions, technical problems, construction work, staff coming too late to their duty, etc. As they are caused by external effects, we cannot control those delays but have to consider them as a part of the input of our optimization strategy. In contrast, delays that are caused by other trains (for example due to waiting for a delayed feeder train or due to conflicts between a delayed train and another train using the same track) or that are caused by a former delay are called *secondary delays*, *follow-up delays*, or *forced delays*. They can (to a limited extend) be influenced by wait/depart decision and priority decisions and are not part of the input.

Within this work, we consider two types of source delays: source delays on events and source delays on activities. A source delay d_i of an event $i \in \mathcal{E}$ might be caused for example by staff coming too late to their duty. In this case, event i cannot take place

at the scheduled time, but later than scheduled. Mathematically, $x_i \geq \pi_i + d_i$ has to be fulfilled (where x_i denotes the time of event $i \in \mathcal{E}$ in the disposition timetable). On the contrary, a source delay d_a of an activity $a = (i, j) \in \mathcal{A}_{\text{train}}$ refers to an activity taking longer than scheduled in the original timetable, caused for example by increased driving time due to bad weather conditions or construction work. A source delay of an activity has to be added to the minimal duration of the activity, i.e. $x_j - x_i \geq L_a + d_a$ has to be satisfied. To simplify the notation, if an event $i \in \mathcal{E}$ has no source delay, we set $d_i = 0$. The same holds for source delays on activities. Given the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, lower bounds $L \in \mathbb{N}^{|\mathcal{A}|}$, a timetable $\pi \in \mathbb{N}^{|\mathcal{E}|}$, and source delays $d \in \mathbb{N}^{|\mathcal{E}|+|\mathcal{A}_{\text{train}}|}$, we are looking for a *disposition timetable* x that at least satisfies

$$\begin{aligned} x_i &\geq \pi_i + d_i && \forall\, i \in \mathcal{E} \\ x_j - x_i &\geq L_a + d_a && \forall\, a = (i, j) \in \mathcal{A}_{\text{train}}. \end{aligned}$$

In practice, more constraints have to be fulfilled – we present an integer programming formulation which also considers changing and headway activities later on in this chapter.

In the following example, we show the difference between both types of source delays. To this end, consider a single train that departs from a station A, arrives at and departs from a subsequent station B, and finally arrives at a third station C. The corresponding event-activity network is depicted in Figure 2.3.

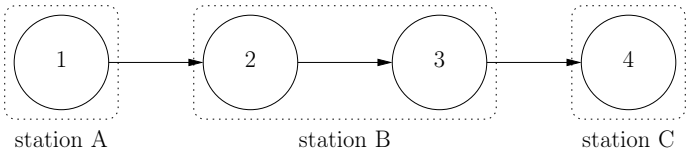

Figure 2.3: The event-activity network for demonstrating the difference between source delays on events and source delays on activities.

We assume minimal driving times $L_{(1,2)} = L_{(3,4)} = 10$, a minimal waiting time of $L_{(2,3)} = 2$ at station B, and a tight schedule with no slack times, given by $\pi_1 = 0$, $\pi_2 = 10$, $\pi_3 = 12$, and $\pi_4 = 22$.

To illustrate the first type of delays, let events 1 and 3 have a source delay of 5 and 3, caused for example by two different drivers who come too late to their duties. When departing from station A, the train has a delay of 5, caused by the delay of the first driver. When arriving at station B, it still has the same delay. Now, the second driver is late, too – however, as the train already has a delay of 5, no additional delay is caused

(as long as the driver's delay is at most 5) and the train departs from station B and arrives at station C with a delay of 5.

To illustrate the second type of delays, assume that not events 1 and 3, but activities (1,2) and (3,4) have a source delay of 5 and 3, caused for example by bad weather conditions. As before, the train arrives with a delay of 5 at station B, and it also leaves station B with a delay of 5. However, while driving from station B to station C, it gets an additional delay of 3, so it arrives with a total delay of 8 at station C. Both scenarios are compared in Table 2.4.

event	1	2	3	4
scheduled time π	0	10	12	22
disposition time if *events* are source-delayed	5	15	17	27
disposition time if *activities* are source-delayed	5	15	17	30

Table 2.4: Source delays on events lead to other delays than source delays on activities.

As can be seen in the example, delays on activities are additive while delays on events are not. In Lemma 2.4, we prove that source delays on events can be replaced by source delays on activities – at the cost of getting a larger event-activity network.

To compute a "good" or an optimal solution of the delay management problem, we assign weights $w_i \in \mathbb{R}^{\geq 0}$ to all events $i \in \mathcal{E}$ and weights $w_a \in \mathbb{R}^{>0}$ to all changing activities $a \in \mathcal{A}_{\text{change}}$. Throughout this work, we assume that w_i is the number of passengers who end their journey with event i (consequently, $w_i = 0$ for all departure events $i \in \mathcal{E}_{\text{dep}}$) and that w_a is the number of passengers who want to use changing activity $a \in \mathcal{A}_{\text{change}}$. In the following, we assume $w_a > 0$ for all activities $a \in \mathcal{A}_{\text{change}}$ (otherwise, nobody would use changing activity a and it therefore could be deleted). However, those weights also could be used to prioritize certain trains (for example high-speed trains) or certain connections (if for example a connection from a long-distance train to a local train is more important than the other way round). Finally, we assume that T is the common period length of all lines (after presenting the model, we discuss why the assumption of a common period of all lines is no severe restriction).

Using those parameters, we can formulate the delay management problem as a linear integer program. To this end, we introduce the following variables to model the disposition timetable and the wait/depart decisions as well as the priority decisions: $x \in \mathbb{N}^{|\mathcal{E}|}$ is the disposition timetable with

$$x_i : \quad \text{time of event } i \in \mathcal{E} \text{ in the disposition timetable.}$$

To model the wait-depart decisions, we use binary variables

$$z_a = \begin{cases} 0 & \text{if changing activity } a \text{ is maintained} \\ 1 & \text{otherwise} \end{cases}$$

for all changing activities $a \in \mathcal{A}_{\text{change}}$. For the priority decisions, we introduce binary variables

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

for all headway activities $(i,j) \in \mathcal{A}_{\text{head}}$. Then, we can model the delay management problem by the following linear integer program:

$$(\mathbf{DM}) \ \min f(x,z,g) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a w_a T \tag{2.1}$$

such that

$$\begin{aligned}
x_i &\geq \pi_i + d_i && \forall\, i \in \mathcal{E} && (2.2)\\
x_j - x_i &\geq L_a + d_a && \forall\, a = (i,j) \in \mathcal{A}_{\text{train}} && (2.3)\\
Mz_a + x_j - x_i &\geq L_a && \forall\, a = (i,j) \in \mathcal{A}_{\text{change}} && (2.4)\\
Mg_{ij} + x_j - x_i &\geq L_{ij} && \forall\, (i,j) \in \mathcal{A}_{\text{head}} && (2.5)\\
g_{ij} + g_{ji} &= 1 && \forall\, (i,j) \in \mathcal{A}_{\text{head}} && (2.6)\\
x_i &\in \mathbb{N} && \forall\, i \in \mathcal{E} && (2.7)\\
z_a &\in \{0,1\} && \forall\, a \in \mathcal{A}_{\text{change}} && (2.8)\\
g_{ij} &\in \{0,1\} && \forall\, (i,j) \in \mathcal{A}_{\text{head}}. && (2.9)
\end{aligned}$$

Here, we assume that M is some constant that is "large enough". In Theorem 3.1 and Corollary 3.2, we show that M indeed can be chosen finitely beforehand. Before analyzing this IP, we shortly explain its meaning:

- The objective (2.1) consists of two parts: the first one is the weighted sum of all delays of all events. As we have chosen the weights w_i to be the number of passengers who end their journey with event $i \in \mathcal{E}$, this is the sum of all delays of those passengers who reach their final destinations with the trains they originally planned to use, without missing a connection. The second part of the objective is the weighted number of missed connections. As we have chosen w_a to be the number of passengers who want to use changing activity $a \in \mathcal{A}_{\text{change}}$, this part is the sum of all passengers who miss a connection, multiplied with the common

period T of all lines. Hence, this term is the additional waiting time for the passengers, caused by missed connections. In general, the objective is an upper bound on the sum of all delays of all passengers at their final destinations; in some cases (if no passenger misses a connection or if the so-called *never-meet property* is fulfilled – see Definition 3.12 and Theorem 3.15), the objective even coincides with that sum. Note that any solution that minimizes (2.1) is a Pareto solution with respect to the two objective functions *minimize the weighted delay over all vehicles* and *minimize the weighted number of missed connections* (the proof is easy: if one could further decrease one term of the objective (2.1) without increasing the other term at the same time, the original solution would not be optimal).

- Constraints (2.2) ensure that no event takes place earlier than scheduled in the original timetable and that source delays on events are respected.
- (2.3): These constraints make sure that the lower bounds of all activities $a \in \mathcal{A}_{\text{train}}$ are respected (as those activities always are fixed) and that source delays on activities are taken into account.
- (2.4): If a connection $a \in \mathcal{A}_{\text{change}}$ is maintained, then $z_a = 0$ and the corresponding constraint ensures that the lower bound on the duration of this activity is respected. However, if a connection is dropped, then $z_a = 1$ and (due to M being "large enough"), (2.4) imposes no additional constraint on the disposition timetable x.
- Constraints (2.5) are similar to constraints (2.4), but for headway activities: If event i should take place before event j, then $g_{ij} = 0$, hence constraints (2.5) impose a constraint on the minimal headway between both events. However, if event j should take place before event i, then $g_{ij} = 1$, and due to M being "large enough", this is no additional constraint on the disposition timetable x.
- (2.6): As headway constraints are disjunctive constraints (either event i has to take place before event j or event j has to take place before event i), these constraints make sure that exactly one headway activity of each pair is selected while the other one is ignored. Note that constraints (2.5) and (2.6) together model the constraints

$$\text{either} \quad x_j - x_i \geq L_{ij} \quad \text{or} \quad x_i - x_j \geq L_{ji}$$

(where "or" is an *exclusive or*). An equivalent formulation is

$$\left| x_j - x_i + \frac{L_{ji} - L_{ij}}{2} \right| \geq \frac{L_{ji} - L_{ij}}{2},$$

see [Sch09b].

- Constraints (2.7) in fact are no additional constraints on an optimal solution if we assume all times π_i in the original timetable and all lower bounds L_a to be integral.

Note that in our model, we make the following assumptions:

- We assume T to be the common period of all lines. This assumption is widespread in the literature. If different lines have different period lengths, this assumption can be relaxed by introducing individual period lengths T_a for all $a \in \mathcal{A}_{\text{change}}$. If, however, we define T as the maximal period length over all lines, then the objective (2.1) overestimates the sum of all delays of all passengers, so by solving (DM) with T "too large", we at least derive an upper bound on the sum of all delays of all passengers in an optimal solution.
- Furthermore, we assume that the routes of the passengers are fixed, i.e. even in the case of delays, a passenger does not use another train than originally planned (except for the case of a missed connection where we assume a passenger to take the same line in the next period). This is discussed in Section 7.4.1 in more detail.

If $\mathcal{A}_{\text{change}} = \emptyset$, the remaining problem is a re-scheduling problem with capacity constraints:

$$(\textbf{Re-Sched}) \ \min f(x,g) = \sum_{i \in \mathcal{E}_{\text{arr}}} w_i(x_i - \pi_i)$$

such that (2.2), (2.3), (2.5)-(2.7), and (2.9) are satisfied.

If we relax all constraints modeling the limited capacity of the tracks (i.e. constraints (2.5), (2.6), and (2.9)), we have the uncapacitated delay management problem:

$$(\textbf{UDM}) \ \min f(x,z) = \sum_{i \in \mathcal{E}_{\text{arr}}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a w_a T$$

such that (2.2)-(2.4), (2.7), and (2.8) are satisfied.

This integer programming formulation of the uncapacitated delay management problem first has been introduced in [Sch01a]. For an in-depth analysis of this model and its relation to other integer programming formulations of the uncapacitated delay management problem, we refer to [Sch06] (we also give a short overview in the end of this chapter). The following relationship between (DM) and (UDM) holds:

Lemma 2.1. *(UDM) is a relaxation of (DM).*

Proof. Each feasible solution of (DM) is feasible for (UDM), and both formulations share the same objective. □

The following lemma is a direct consequence of Lemma 2.1:

Lemma 2.2. *Let F^{DM} and F^{UDM} denote the objective value of the optimal solution of (DM) and the objective value of the optimal solution of the corresponding instance of (UDM), respectively. Then*

$$F^{\mathrm{UDM}} \leq F^{\mathrm{DM}}.$$

When introducing heuristic solution approaches for solving the capacitated delay management problem in Chapter 4, we fix the priority decisions heuristically and treat the corresponding fixed headway activities like the fixed activities in $\mathcal{A}_{\mathrm{train}}$. For some set $\mathcal{A}_{\mathrm{fix}} \subset \mathcal{A}_{\mathrm{head}}$, we hence define

$$(\mathbf{UDM}(\mathcal{A}_{\mathrm{fix}}))\ \min f(x,z) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\mathrm{change}}} z_a w_a T$$

such that

$$x_j - x_i \geq L_a \quad \forall a = (i,j) \in \mathcal{A}_{\mathrm{fix}} \tag{2.10}$$

and such that (2.2)-(2.4), (2.7), and (2.8) are satisfied.

Note that UDM($\mathcal{A}_{\mathrm{fix}}$) is only feasible if the corresponding event-activity network does not contain any directed cycle, i.e. if fixing the priority decisions is done in a reasonable way. In this case, we obtain an upper bound on the optimal solution of the capacitated delay management problem:

Lemma 2.3. *Let $\mathcal{A}_{\mathrm{fix}} = \{(i,j) \in \mathcal{A}_{\mathrm{head}} : g_{ij} = 0\}$ for some $g \in \{0,1\}^{|\mathcal{A}_{\mathrm{head}}|}$ which satisfies* (2.6) *and assume that UDM($\mathcal{A}_{\mathrm{fix}}$) is feasible. Then,*

$$F^{\mathrm{DM}} \leq F^{\mathrm{UDM}(\mathcal{A}_{\mathrm{fix}})}.$$

Some of the solution approaches which we suggest in Chapter 4 do not only fix the priority decisions, but also the wait/depart decisions heuristically. In this case, we obtain a set $\mathcal{A}_{\mathrm{fix}} := \{a \in \mathcal{A}_{\mathrm{change}} : z_a = 0\ \} \cup \{(i,j) \in \mathcal{A}_{\mathrm{head}} : g_{ij} = 0\}$. Determining the remaining variables x_i in (DM) then reduces to a simple project planning problem:

$$(\mathbf{PP}(\mathcal{A}_{\mathrm{fix}}))\ \min f(x) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i)$$

such that (2.2), (2.3), (2.7), and (2.10) are satisfied.

Again (PP($\mathcal{A}_{\text{fix}}$)) is only feasible if the corresponding event-activity network does not contain any directed cycle. (PP($\mathcal{A}_{\text{fix}}$)) can be solved in polynomial time, for example by applying the forward phase of the Critical Path Method (Cpm) from project planning (see [LTW63, Elm77]) as follows:

Algorithm 2.1: Critical Path Method (Cpm)

Input: An event-activity network $\mathcal{N} = \{\mathcal{E}, \mathcal{A}\}$ with lower bounds $L \in \mathbb{N}^{|\mathcal{A}|}$, a set $\mathcal{A}_{\text{fix}} \subset \mathcal{A}$, a timetable $\pi \in \mathbb{N}^{|\mathcal{E}|}$, and source delays $d \in \mathbb{N}^{|\mathcal{E}|+|\mathcal{A}_{\text{train}}|}$.

Step 1: Sort $(\mathcal{E}, \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{fix}})$ topologically and obtain $\mathcal{E} = \{i_1, \ldots, i_{|\mathcal{E}|}\}$.

Step 2: Set $x_{i_1} := \pi_{i_1} + d_{i_1}$.

Step 3: Iteratively, set $x_k := \max \left\{ \pi_k + d_k, \max_{a=(i,k) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{fix}}} x_i + L_a + d_a \right\}$ for all $k \in \{i_2, \ldots, i_{|\mathcal{E}|}\}$.

Output: A disposition timetable $x \in \mathbb{N}^{|\mathcal{E}|}$.

Usually, $\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{head}}$. However, for delay management with integrated rolling stock circulations (see Chapter 5), $\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{head}} \cup \mathcal{A}_{\text{circ}}$.

The first step of Cpm can be done in time $\Theta(|\mathcal{E}| + |\mathcal{A}|)$ by applying a depth-first search which also detects whether the graph is acyclic or not (i.e. whether (PP($\mathcal{A}_{\text{fix}}$)) is feasible), see [CLRS01]. Step 2 and 3 together have a runtime of $\Theta(|\mathcal{E}| + |\mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{fix}}|)$ (as each event is considered once and each activity enters exactly one event). Hence, Cpm has linear worst-case runtime of $\mathcal{O}(|\mathcal{E}| + |\mathcal{A}|)$.

Note that Cpm also can be used to compute an initial (nonperiodic) timetable: Set $\pi_i = d_i = 0$ for all events $i \in \mathcal{E}$ and $d_a = 0$ for all activities $a \in \mathcal{A}$, then apply Cpm. By doing so, we can use Cpm for example to compute robust timetables in the context of recoverable robustness, see Chapter 6.

In the following, we call a disposition timetable x *time-minimal* if for all disposition timetables x' that are feasible for fixed $z \in \{0,1\}^{|\mathcal{A}_{\text{change}}|}$ and $g \in \{0,1\}^{|\mathcal{A}_{\text{head}}|}$, $x_i \leq x'_i$ for all $i \in \mathcal{E}$. This extends the definition of time-minimal given in [Sch06]. By construction, Cpm always computes a time-minimal solution.

As already mentioned, the objective (2.1) in general only is an upper bound on the sum of all delays of all passengers at their final destinations and coincides with this sum only in some special cases. To minimize the actual delays in *all* cases, in [Sch06], different

integer programming formulations for solving the uncapacitated delay management problem are suggested. The starting point is a path-oriented model where the path each passenger uses in the event-activity network is taken into account. While in (DM), we consider wait/depart decisions based on connections (one decision variable z_a for each connection a), in the first approach suggested in [Sch06], wait/depart decisions are made on the basis of the passengers' paths (one decision variable z_p for each path p). Then, the objective of the corresponding IP formulation is

$$\min \sum_{p \in P} w_p \left[(x_{i(p)} - \pi_{i(p)})(1 - z_p) + T z_p \right] \tag{2.11}$$

where P denotes the set of all passenger paths, w_p is the number of passengers using path p, $i(p)$ denotes the last event on path p, and the binary decision variable z_p is 0 if and only if all connections on path p are maintained. From the point of view of a human dispatcher, the activity-oriented approach is much more natural since a dispatcher has to decide whether a *connection* should be maintained, not whether a passenger's *path* should be maintained. The resulting integer program (TDM-A) has the advantage of minimizing the exact sum of all delays of all passengers at their final destinations in all cases. However, it has the drawback of having a quadratic objective function. A linearization of (TDM-A) is possible and yields an integer programming formulation which is called (TDM-B). However, (TDM-B) is a significantly weaker formulation than (TDM-A). Furthermore, both models do not allow to drop the assumption of a common period of all lines. To circumvent these disadvantages, a third model (which focuses on the activities, not on the passenger paths) is suggested. The resulting integer programming formulation (TDM-C) is cubic and contains much more variables and constraints – however, it allows to drop the assumption of a common period T and is a stronger formulation than the others. Furthermore, it can be linearized, yielding an integer program that minimizes the sum of all delays of all passengers at their final destinations whenever the never-meet property is fulfilled. This linear formulation (TDM-const) is equivalent to (UDM).

As already mentioned before, it is possible to transform a problem with source-delayed events into a problem with only source-delayed activities by introducing a virtual event e_0 and virtual activities $a_i := (e_0, i)$ with $d_{a_i} := d_i$ for each source delayed event i (for a demonstration, see Figure 2.5 and Figure 2.6).

To prove this claim, we need the following notation: An *instance* i of problem (DM) is given by a tuple $i = (\mathcal{N}, L, w, \pi, d)$ where $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is the event-activity network of the instance, $L \in \mathbb{N}^{|\mathcal{A}|}$ are the lower bounds of all activities, $w \in \mathbb{R}^{|\mathcal{E}|+|\mathcal{A}_{\text{change}}|}$ are the passenger weights on events and changing activities, $\pi \in \mathbb{N}^{|\mathcal{E}|}$ is a feasible timetable, and $d \in \mathbb{N}^{|\mathcal{E}|+|\mathcal{A}_{\text{train}}|}$ are the source delays.

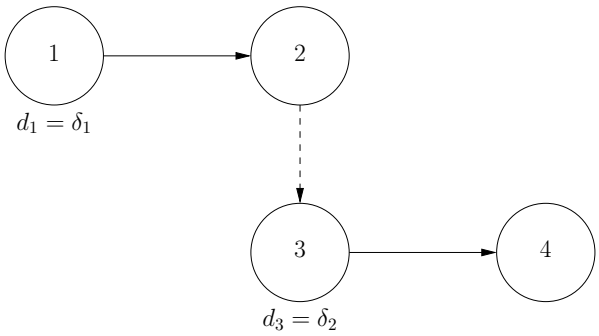

Figure 2.5: An event-activity network with source delays on events.

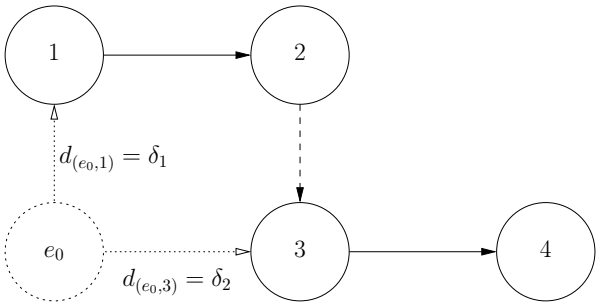

Figure 2.6: The corresponding event-activity network where source delays on events have been replaced by source delays on ("virtual") activities.

By iteratively applying the following construction, we can transfer each instance with source-delayed events to an instance with source delays only on activities:

Lemma 2.4. *Given an instance $i = (\mathcal{N}, L, w, \pi, d)$ of (DM) with $d_k > 0$ for some $k \in \mathcal{E}$, we define the instance $\tilde{i} = (\tilde{\mathcal{N}}, \tilde{L}, \tilde{w}, \tilde{\pi}, \tilde{d})$ by adding an event e_0 with $\tilde{w}_{e_0} := 1$, $\tilde{\pi}_{e_0} := \pi_k$, $\tilde{d}_{e_0} := 0$ and an activity (e_0, k) with $\tilde{L}_{(e_0,k)} := 0$ and $\tilde{d}_{(e_0,k)} := d_k$. Furthermore, we change d_k to $\tilde{d}_k := 0$. Then, each optimal solution for the instance $(\tilde{\mathcal{N}}, \tilde{L}, \tilde{w}, \tilde{\pi}, \tilde{d})$ also is an optimal solution for the initial instance $(\mathcal{N}, L, w, \pi, d)$ and vice versa.*

Proof. In each optimal solution for the modified instance, event e_0 takes place as early as possible since its weight is strictly positive. As it has no incoming activities and no source delay, $x_{e_0} = \pi_{e_0} = \pi_k$ in each optimal solution. In the IP corresponding to the modified instance, the constraint $x_k \geq \pi_k + d_k$ from the IP corresponding to the original instance is changed to $x_k \geq \pi_k + \tilde{d}_k = \pi_k$, and we have the additional constraint $x_k - x_{e_0} \geq \tilde{L}_{(e_0,k)} + \tilde{d}_{(e_0,k)}$ which is equivalent to $x_k \geq x_{e_0} + \tilde{L}_{(e_0,k)} + \tilde{d}_{(e_0,k)} = \pi_k + 0 + d_k$. Both constraints together are equivalent to the constraint $x_k \geq \pi_k + d_k$ from the IP

corresponding to the original instance, and as all other constraints are the same for both instances, this finishes the proof. □

Since replacing source delays on events by source delays on activities enlarges the event-activity network and makes it harder to prove tight bounds on the constant M (see Chapter 3), we do not follow this approach here but treat both types of delays throughout this work.

Chapter 3

Analyzing the Model

In this chapter, we further investigate the integer programming formulation (DM) presented in Chapter 2 and analyze the headway constraints. In Section 3.1, we show that the constant M can be chosen finitely beforehand. We then modify the IP formulation in Section 3.2 to better take into account practical needs from real-world applications. This modification allows us to tighten the bound on M derived in Section 3.1 and yields an approach for reducing the number of headway constraints significantly. In Section 3.3, we show that – in an optimal solution – backward headways never carry over a delay to a non-delayed event and use this knowledge to suggest another preprocessing step for reducing the input size of the instance of (DM). We also use this result to extend some properties of the uncapacitated delay management problem that have been proven in [Sch06] to the capacitated case in Section 3.4. Finally, in Section 3.5, we present some numerical results from a case study based on real-world data, demonstrating the effectiveness of our reduction techniques. Parts of Sections 3.1, 3.3, and 3.4 are results from a joint research published in [SS08], see also [SS10].

3.1 Analyzing the IP

In this section, we prove that the constant M in the IP formulation (2.1)-(2.9) indeed can be chosen finitely beforehand, depending on the input instance. This is not only important as it proves that the IP formulation is well-defined; it is also important for one other reason: if M is chosen too small, then feasible or even optimal solutions are cut off. However, if M is rather large, then due to the fact that commercial IP solvers solve the problem numerically instead of exactly, an infeasible solution might be accepted as feasible. Hence, it is desirable to prove a *tight* bound on M to choose

M as large as necessary, but as small as possible. Thus, in Section 3.2, we suggest a modification of the IP formulation (DM) which allows to dramatically decrease the value of M. Before proving a first bound, we demonstrate both effects by a small example with three stations A, B, and C.

First, we show the effect of choosing M too small. Assume that one train is driving from station A to station B while another train is driving from station B to station C. Within station B, passengers might transfer from the first train to the second one. See Figure 3.1 for an illustration of the corresponding event-activity network.

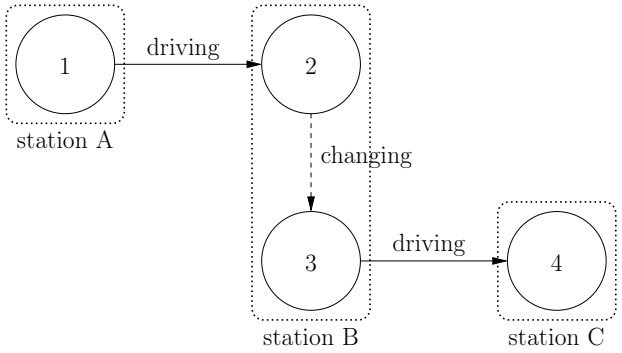

Figure 3.1: Example for showing that choosing M too small cuts off optimal solutions.

Assume that all activities have a lower bound of 1, i.e. $L_{(1,2)} = L_{(2,3)} = L_{(3,4)} = 1$. Furthermore, assume weights $w_1 = w_3 = 0$, $w_2 = 1$, $w_4 = 3$, and $w_{(2,3)} = 1$, a common period $T = 10$, and a source delay $d_1 = 5$ of the first event.

Depending on whether we decide to maintain or not to maintain the connection, we obtain the following time-minimal disposition timetables:

event i	1	2	3	4
π_i	0	1	2	3
$x_i^{\text{no-wait}}$	5	6	2	3
x_i^{wait}	5	6	7	8

The corresponding objective values of both solutions are $F^{\text{wait}} = 5w_2 + 5w_4 = 20$ and $F^{\text{no-wait}} = 5w_2 + Tw_{(2,3)} = 15$. Hence $(x^{\text{no-wait}}, z_{(2,3)} = 1)$ is the optimal solution. If we use $M = 5$, then this optimal solution of the delay management problem is feasible for the IP formulation (DM), i.e. M is large enough – this is proven in Corollary 3.2 and can also be verified quickly by checking constraint (2.5).

If, however, we use $M = 4$, then $(x^{\text{no-wait}}, z_{(2,3)} = 1)$ no longer is feasible for (DM) as

$$Mz_{(2,3)} + x_3 - x_2 = 4 + 2 - 6 = 0 < 1 = L_{(2,3)},$$

hence constraint (2.5) is violated. To "repair" the solution, for fixed $z_{(2,3)} = 1$, x_3 and x_4 have to be increased by $k \geq 1$ as

$$Mz_{(2,3)} + (x_3 + k) - x_2 \geq 4 + 3 - 6 = 1 = L_{(2,3)},$$

leading to an increase of the objective value of $k \cdot w_4$. For $k = 1$, the solution is optimal with objective value 18 as it is time-minimal. Switching to $z_{(2,3)} = 0$ is valid only for $k \geq 5$, leading to an objective value of $5w_2 + k \cdot w_4 = 5 + 3k \geq 20 > 18$. Hence $z_{(2,3)} = 1$ still is the optimal wait/depart decision, but the disposition times of events 3 and 4 have to be increased by 1, compared to the optimal solution where M is large enough.

If we reduce M even further, i.e. $M = 3$, then also the solution for $M = 4$ is not feasible anymore for (DM). In this case, the optimal solution is to further increase x_3 and x_4 by 4, compared to the previous solution for $M = 4$, and to maintain the connection, i.e. to set $z_{(2,3)} = 0$.

Summing up, we have the following optimal solutions for the IP, depending on M:

	x_1	x_2	x_3	x_4	$z_{(2,3)}$	$f(x, z)$
$M = 5$	5	6	2	3	1	15
$M = 4$	5	6	3	4	1	18
$M = 3$	5	6	7	8	0	20

So in conclusion, it is important not to choose M too small – otherwise, optimal solutions might be cut off.

However, a large value of M might result in a wrong objective value or even an infeasible solution that is accepted as feasible by the IP solver. To demonstrate the first effect, we use the same example as above. We assume that the numerical tolerance how much an integer variable is allowed to differ from the nearest integer value is $5 \cdot 10^{-6}$ (which is the default value of the control parameter MIPTOL in FICO Xpress 7) and that $M = 10^6$. Then the optimal solution of the resulting IP for the example introduced above that is computed by the solver is $z_{(2,3)} = 5 \cdot 10^{-6}$ (which is recognized as the integer value 0 by the solver, due to the numerical tolerance – hence, for the solver, constraint (2.8) is fulfilled) and $x_1 = 5$, $x_2 = 6$, $x_3 = 2$, $x_4 = 3$ (which is feasible as $Mz_{(2,3)} + x_3 - x_2 = 10^6 \cdot 5 \cdot 10^{-6} + 2 - 6 = 1 = L_{(2,3)}$, i.e. constraint (2.4) is satisfied)

with objective value $5w_2 + Tz_a w_{(2,3)} = 5 + 5 \cdot 10^{-5}$. In this case, the disposition timetable x is feasible for (DM) only if $z_{(2,3)} = 1$ (which yields an objective value of 15). Hence in this example, the value of $z_{(2,3)}$ in the solution is wrong, but at least all operational constraints are satisfied.

If headways are involved, a large value of M can even lead to an infeasible disposition timetable that violates operational constraints like minimum security distances modeled by headway activities. To demonstrate this, we again assume that the tolerance how much an integer variable is allowed to differ from the nearest integer value is $5 \cdot 10^{-6}$ and assume $M = 2 \cdot 10^7$ (note that for the data of our case study in Section 3.5, depending on the observation period and the actual source delays, the bound for M resulting from Theorem 3.1 ranges from about $1.5 \cdot 10^6$ to about $2.5 \cdot 10^8$, i.e. the assumption $M = 2 \cdot 10^7$ is not unrealistic). Then the constraint $Mg_{ij} + x_j - x_i \geq L_{ij}$ is satisfied for all $x_j \geq x_i + (L_{ij} - 100)$ if $g_{ij} = 5 \cdot 10^{-6}$ (which, due to the tolerance, is accepted as zero by the solver), i.e. the lower bound of the headway activity (i, j) might be violated by up to 100.

In conclusion, choosing a too small value for M can "cut off" optimal solutions, leading to a degradation of the objective value, or even make the problem infeasible (if headway constraints are affected). On the other hand, a large value of M might yield infeasible solutions that are accepted as feasible by the solver, due to rounding errors and due to the limited numerical accuracy.

As a consequence, we need a tight bound on M. To this end, in Theorem 3.1, we give an upper bound on the delay of a single event in an optimal solution and use this knowledge to prove an upper bound on the constant M in Corollary 3.2. In Section 3.2, we then modify the problem (DM) to tighten the bound on M.

Theorem 3.1. *Let an instance of (DM) be given and let*

$$D := \max_{i \in \mathcal{E}} d_i + \sum_{a \in \mathcal{A}_{\text{train}}} d_a + \sum_{(i,j) \in \mathcal{A}^{\text{back}}_{\text{head}}} \pi_i - \pi_j + L_{ij}. \tag{3.1}$$

Then there exists an optimal solution (x, z, g) of (DM) such that $x_k \leq \pi_k + D$ for all events $k \in \mathcal{E}$.

Proof. We show the following stronger statement: For any feasible solution $(\bar{x}, z, g)$ of (DM), there exists a feasible solution $(\tilde{x}, z, g)$ with $\tilde{x}_k \leq \bar{x}_k$ for all events $k \in \mathcal{E}$ (hence $f(\tilde{x}, z, g) \leq f(\bar{x}, z, g)$) that satisfies $\tilde{x}_k \leq \pi_k + D$ for all $k \in \mathcal{E}$. The claim of the theorem then is a direct consequence of this stronger statement.

Given a solution $(\bar{x}, z, g)$ of (DM), we define

$$\begin{aligned}
\mathcal{A}^{\text{fix}}_{\text{change}} &:= \{a \in \mathcal{A}_{\text{change}} : z_a = 0\}, \\
\mathcal{A}^{\text{fix}}_{\text{head}} &:= \{(i,j) \in \mathcal{A}_{\text{head}} : g_{ij} = 0\}, \\
\mathcal{A}^{\text{fix-bw}}_{\text{head}} &:= \mathcal{A}^{\text{fix}}_{\text{head}} \cap \mathcal{A}^{\text{back}}_{\text{head}}, \\
\mathcal{A}^{\text{fix-fw}}_{\text{head}} &:= \mathcal{A}^{\text{fix}}_{\text{head}} \cap \mathcal{A}^{\text{forw}}_{\text{head}}, \\
\mathcal{A}_{\text{fix}} &:= \mathcal{A}^{\text{fix}}_{\text{change}} \cup \mathcal{A}^{\text{fix}}_{\text{head}}.
\end{aligned}$$

As $(\bar{x}, z, g)$ is a feasible solution, $\mathcal{N}' := (\mathcal{E}, \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{fix}})$ is acyclic, so we can use algorithm CPM with $\mathcal{A}_{\text{fix}}$ as defined above to compute a disposition timetable $\tilde{x}$. By $\prec$, we denote the order on the set $\mathcal{E} = \{i_1, \ldots, i_{|\mathcal{E}|}\}$ of events that is gained by sorting the graph topologically. As CPM computes a time-minimal solution, $\tilde{x}$ satisfies $\tilde{x}_k \leq \bar{x}_k$. Then, we can inductively prove the following bound on the delay of an event $k \in \mathcal{E}$ in the disposition timetable $\tilde{x}$ that only depends on k's predecessors:

Claim: For each $k \in \mathcal{E}$, we have $\tilde{x}_k \leq \pi_k + U_k$ with

$$U_k = \max_{\substack{i \in \mathcal{E}: \\ i \preceq k}} d_i + \sum_{\substack{a=(i,j) \in \mathcal{A}_{\text{train}}: \\ j \preceq k}} d_a + \sum_{\substack{(i,j) \in \mathcal{A}^{\text{fix-bw}}_{\text{head}}: \\ j \preceq k}} \pi_i - \pi_j + L_{ij}.$$

We prove the claim by induction.

Basis: For the first event i_1, according to the definition of algorithm CPM, we have $\tilde{x}_{i_1} = \pi_{i_1} + d_{i_1} \leq \pi_{i_1} + U_{i_1}$, so the claim is true for i_1.

Inductive step: Let $k \in \{i_2, \ldots, i_{|\mathcal{E}|}\}$ and assume that the claim already has been proven for all events $i \prec k$. We distinguish the two following cases, depending on which term in Step 3 of algorithm CPM is maximal when computing $\tilde{x}_k$:

1. $\tilde{x}_k = \pi_k + d_k$. Since $d_k \leq U_k$, the claim is true.
2. $\tilde{x}_k = \tilde{x}_i + L_a + d_a$ for some $a = (i,k) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{fix}}$. Depending on the type of activity a, we distinguish the following two cases:
   a) If $a = (i,k) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}^{\text{fix}}_{\text{change}} \cup \mathcal{A}^{\text{fix-fw}}_{\text{head}}$, then

$$\begin{aligned}
\tilde{x}_k &= \tilde{x}_i + L_a + d_a \\
&\leq \pi_i + U_i + L_a + d_a \\
&= \pi_i + L_a + \max_{\substack{i' \in \mathcal{E}: \\ i' \preceq i}} d_{i'} + d_a + \sum_{\substack{a'=(i',j) \in \mathcal{A}_{\text{train}}: \\ j \preceq i}} d_{a'} + \sum_{\substack{(i',j) \in \mathcal{A}^{\text{fix-bw}}_{\text{head}} \\ j \preceq i}} \pi_{i'} - \pi_j + L_{i'j} \\
&\leq \pi_k + \max_{\substack{i' \in \mathcal{E}: \\ i' \preceq i}} d_{i'} + d_a + \sum_{\substack{a'=(i',j) \in \mathcal{A}_{\text{train}}: \\ j \preceq i}} d_{a'} + \sum_{\substack{(i',j) \in \mathcal{A}^{\text{fix-bw}}_{\text{head}} \\ j \preceq i}} \pi_{i'} - \pi_j + L_{i'j}
\end{aligned}$$

where the second line is a direct consequence of the induction hypothesis for event $i \prec k$. The last inequality holds as π is a feasible timetable that satisfies $\pi_k \geq \pi_i + L_a$ for all $a = (i,k) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{head}}^{\text{forw}}$. The claim then follows by moving d_a (which is equal to 0 for $a \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}$) to the first sum and using the transitivity of the order $\prec$.

b) If $a = (i,k) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}$, we obtain

$$\begin{aligned}\tilde{x}_k &= \tilde{x}_i + L_{ik} \\ &\leq \pi_i + U_i + L_{ik} \\ &= \quad \pi_k + \pi_i - \pi_k + L_{ik} \\ &\quad + \max_{\substack{i' \in \mathcal{E}: \\ i' \preceq i}} d_{i'} + \sum_{\substack{a'=(i',j) \in \mathcal{A}_{\text{train}}: \\ j \prec i}} d_{a'} + \sum_{\substack{(i',j) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}} \\ j \prec i}} \pi_{i'} - \pi_j + L_{i'j},\end{aligned}$$

and the claim follows by moving $\pi_i - \pi_k + L_{ik}$ to the second sum and again using the transitivity of the order $\prec$.

As $U_k \leq D$ for all $k \in \mathcal{E}$, the theorem holds. □

Using this result, we can give an upper bound on the minimal size needed for M in the IP formulation (DM):

Corollary 3.2. *$M \geq D$ is "large enough".*

Proof. Let $M \geq D$ and let (x, z, g) be an optimal solution of the delay management problem that satisfies constraints (2.2), (2.3), and (2.6)-(2.9) of the IP formulation (DM) as well as (2.4) for all $a \in \mathcal{A}_{\text{change}}$ with $z_a = 0$ and (2.5) for all $(i,j) \in \mathcal{A}_{\text{head}}$ with $g_{ij} = 0$. We have to show that x then also satisfies (2.4) for all $a \in \mathcal{A}_{\text{change}}$ with $z_a = 1$ and (2.5) for all $(i,j) \in \mathcal{A}_{\text{head}}$ with $g_{ij} = 1$ (i.e. that, due to the size of M, these constraints are fulfilled "automatically" for all dropped connections and for all dropped headways).

First, let $a = (i,j) \in \mathcal{A}_{\text{change}}$ with $z_a = 1$. We have to show that constraint (2.4) is satisfied, i.e. that $M + x_j - x_i \geq L_a$ holds. From Theorem 3.1, we know that $x_i \leq \pi_i + D$. As π is a feasible timetable, $\pi_j - \pi_i \geq L_a$, and as x satisfies constraints (2.2), we have $x_j \geq \pi_j + d_j \geq \pi_j$. Hence,

$$\begin{aligned}M &\geq D \\ &\geq x_i - \pi_i \\ &\geq x_i - \pi_j + L_a \\ &\geq x_i - x_j + L_a,\end{aligned}$$

thus $M + x_j - x_i \geq L_a$, so (2.4) indeed is satisfied for *all* $a \in \mathcal{A}_{\text{change}}$, including all dropped connections.

Now, let $a = (i, j) \in \mathcal{A}_{\text{head}}$ with $g_{ij} = 0$. We have to show that the corresponding (dropped) headway $a^{-1} = (j, i)$ satisfies constraint (2.5), i.e. $M + x_i - x_j \geq L_{ji}$. We distinguish two cases, depending on whether a is a forward headway or a backward headway.

1. Let $a \in \mathcal{A}^{\text{forw}}_{\text{head}}$. First, from the proof of Theorem 3.1, we know $x_j \leq \pi_j + U_j$. Secondly, according to the definition of U_j and D, each term added to U_j also is added to D, but not the other way round: As $a = (i, j) \in \mathcal{A}^{\text{forw}}_{\text{head}}$, for $a^{-1} = (j, i)$, we have $a^{-1} \in \mathcal{A}^{\text{back}}_{\text{head}}$, but $a^{-1} \notin \mathcal{A}^{\text{fix-bw}}_{\text{head}}$ (as already a is fixed). Hence the term $\pi_j - \pi_i + L_{ji}$ is included in D, but not in U_j. Thus, $D - U_j \geq \pi_j - \pi_i + L_{ji}$. Last, as x satisfies (2.2), $x_i \geq \pi_i + d_i \geq \pi_i$. Putting all together yields

$$\begin{aligned} x_j &\leq \pi_j + U_j \\ &\leq \pi_j + D - (\pi_j - \pi_i + L_{ji}) \\ &= D + \pi_i - L_{ji} \\ &\leq D + x_i - L_{ji}. \end{aligned}$$

   As $M \geq D$, this yields $M + x_i - x_j \geq L_{ji}$.

2. Let $a \in \mathcal{A}^{\text{back}}_{\text{head}}$. From Theorem 3.1, we know that $x_j \leq \pi_j + D$. As π is a feasible timetable, $\pi_i > \pi_j$ implies $\pi_i - \pi_j \geq L_{ji}$. As x satisfies constraints (2.2), we have $x_i \geq \pi_i + d_i \geq \pi_i$. Putting all together,

$$\begin{aligned} M &\geq D \\ &\geq x_j - \pi_j \\ &\geq x_j - \pi_i + L_{ji} \\ &\geq x_j - x_i + L_{ji}, \end{aligned}$$

   hence again we have $M + x_i - x_j \geq L_{ji}$.

Both cases show that (2.5) indeed is satisfied for *all* $(i, j) \in \mathcal{A}_{\text{head}}$, including the dropped headways. □

Compared to (UDM) where

$$M = \max_{i \in \mathcal{E}} d_i + \sum_{a \in \mathcal{A}_{\text{train}}} d_a$$

is large enough, the constant needed for (DM) is rather large, yielding a weak linear programming relaxation of the integer program.

In Section 3.3, we further investigate the backward headways and show how the integer programming formulation can be improved. One result shows that the bound on M can be tightened – but first we analyze a slight modification of (DM) that nevertheless provides a significant reduction of the bound on M (at the cost of introducing additional restrictions on the disposition timetable).

3.2 Bounding the Maximal Delay

In practice, a reasonable constraint on a disposition timetable is that the delay of each event does not get "too large". By imposing such a constraint, we might loose some optimal solutions of (DM) where a single train with low passenger weights has a very large delay, and if we set the bound on the maximal delay too low, we might even end up with an infeasible problem. However, if we restrict the maximum delay in a reasonable way, we can significantly reduce the number of headway constraints which we have to consider, and we can drastically reduce the size of the constant M in the IP, compared to the quite large value shown to be "large enough" in Theorem 3.1 and Corollary 3.2.

To this end, let $Y \in \mathbb{N}$ be the maximal delay which we want to allow for a single event. Then, for each event, we add one constraint to the IP formulation (2.1)-(2.9) of (DM) to bound the maximal delay and obtain the subproblem which we call bounded delay management problem (BDM):

$$(\mathbf{BDM}) \ \min f(x,z,g) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a w_a T$$

such that

$$x_i \leq \pi_i + Y \quad \forall i \in \mathcal{E} \tag{3.2}$$

and such that (2.2)-(2.9) are satisfied.

The following relationship between (DM) and (BDM) holds:

Lemma 3.3. *(DM) is a relaxation of (BDM).*

Proof. Each feasible solution of (BDM) is also a feasible solution of (DM), and the objective functions of both problems are identically. □

The following Lemma is a consequence of Lemma 3.3:

Lemma 3.4. *Let F^{BDM} and F^{DM} denote the objective value of the optimal solution of (BDM) and the objective value of the optimal solution of the corresponding instance of (DM), respectively. Then*

$$F^{\mathrm{DM}} \leq F^{\mathrm{BDM}}.$$

For problem (BDM), we can tighten the bound on M from Corollary 3.2 significantly:

Theorem 3.5. *Given an instance of (BDM),*

$$M := Y + \max_{(i,j)\in\mathcal{A}_{\mathrm{head}}} (\pi_j - \pi_i + L_{ji}) \tag{3.3}$$

is "large enough".

Proof. Let M be defined as in (3.3). The proof is similar to the proof of Corollary 3.2: Let (x, z, g) be a solution of the delay management problem with bounded delay that satisfies constraints (2.2), (3.2), (2.3), and (2.6)-(2.9) as well as (2.4) for all $a \in \mathcal{A}_{\mathrm{change}}$ with $z_a = 0$ and (2.5) for all $(i, j) \in \mathcal{A}_{\mathrm{head}}$ with $g_{ij} = 0$. We have to show that x then also satisfies (2.4) for all $a \in \mathcal{A}_{\mathrm{change}}$ with $z_a = 1$ and (2.5) for all $(i, j) \in \mathcal{A}_{\mathrm{head}}$ with $g_{ij} = 1$ (i.e. that, due to the size of M, these constraints are fulfilled "automatically" for all dropped connections and for all dropped headways).

First, let $a = (i, j) \in \mathcal{A}_{\mathrm{change}}$ with $z_a = 1$. Then, using $x_j \geq \pi_j$ and $x_i \leq \pi_i + Y$ (as x satisfies (2.2) and (3.2)) as well as $\pi_j - \pi_i \geq L_a$ (as π is a feasible timetable and hence satisfies (2.4)), we have

$$\begin{aligned}
Mz_a + x_j - x_i &= M + x_j - x_i \\
&= Y + \max_{(i,j)\in\mathcal{A}_{\mathrm{head}}} (\pi_j - \pi_i + L_{ji}) + x_j - x_i \\
&\geq Y + x_j - x_i \\
&\geq Y + \pi_j - (\pi_i + Y) \\
&= \pi_j - \pi_i \\
&\geq L_a.
\end{aligned}$$

Now, let $(i, j) \in \mathcal{A}_{\mathrm{head}}$ with $g_{ij} = 1$. Then,

$$\begin{aligned}
Mg_{ij} + x_j - x_i &= M + x_j - x_i \\
&= Y + \max_{(i,j)\in\mathcal{A}_{\mathrm{head}}} (\pi_j - \pi_i + L_{ji}) + x_j - x_i \\
&\geq Y + (\pi_i - \pi_j + L_{ij}) + x_j - x_i \\
&= Y + (\pi_i - x_i) + (x_j - \pi_j) + L_{ij} \\
&\geq Y + (\pi_i - (\pi_i + Y)) + 0 + L_{ij} \\
&= L_{ij}.
\end{aligned}$$

Hence the claim is true. □

Note that for large instances with many headway activities and for a "reasonable" Y,

$$Y + \max_{(i,j)\in\mathcal{A}_{\text{head}}} (\pi_j - \pi_i + L_{ji}) \;\ll\; \max_{i\in\mathcal{E}} d_i + \sum_{a\in\mathcal{A}_{\text{train}}} d_a + \sum_{(i,j)\in\mathcal{A}_{\text{head}}^{\text{back}}} \pi_i - \pi_j + L_{ij},$$

so (BDM) might have a much stronger linear programming relaxation than (DM), depending on the input instance and the choice of Y. In addition, a significantly smaller value for M reduces those rounding errors and numerical instabilities that we described in Section 3.1. In our numerical experiments (see Section 3.5), M as defined in (3.3) was between 800 and 15 000 times smaller than M as defined in (3.1).

Apart from Theorem 3.5, bounding the maximal delay has another advantage – it can be used to fix some priority decisions before solving the problem:

Theorem 3.6. *Given an instance of (BDM) with $Y \in \mathbb{N}$, assume that $\pi_j - \pi_i > Y$ for some $(i,j) \in \mathcal{A}_{\text{head}}$. Then, in each feasible solution of (BDM), $g_{ij} = 0$, i.e. event i is scheduled first.*

Proof. By contradiction. For a given instance of (BDM), let (x, z, g) be a feasible solution with $g_{ij} = 1$ and $\pi_j - \pi_i > Y$ for some $(i,j) \in \mathcal{A}_{\text{head}}$. As $g_{ij} = 1$ and (x, z, g) is a feasible solution satisfying (2.5), we have $x_i - x_j \geq L_{ji}$. Hence

$$\begin{aligned} x_i &\geq x_j + L_{ji} \\ &\geq \pi_j + L_{ji} \\ &> \pi_i + Y + L_{ji}, \end{aligned}$$

thus constraint (3.2) is violated and (x, z, g) is not feasible. □

The statement of Theorem 3.6 can be used to significantly reduce the number of priority decisions:

- For each $(i,j) \in \mathcal{A}_{\text{head}}$ with $\pi_j - \pi_i > Y$, we can fix $g_{ij} = 0$ and $g_{ji} = 1$, i.e. we can delete the backward headway (j,i) and treat the forward headway (i,j) like the fixed activities in $\mathcal{A}_{\text{train}}$.
- For each $(i,j) \in \mathcal{A}_{\text{head}}$ with $\pi_j - \pi_i \geq Y + L_{ij}$, we can even completely delete *both* headways (i,j) and (j,i).

The latter is a consequence of

$$x_j \geq \pi_j \geq \pi_i + Y + L_{ij} \geq x_i + L_{ij},$$

i.e. due to the fact that we have bounded x_i by $\pi_i + Y$, all headway activities (i,j) where $\pi_j - \pi_i$ is large enough are respected "automatically".

We formalize this result in the following algorithm Fix-Headways:

Algorithm 3.1: Fix-Headways

Input: An event-activity network $\mathcal{N}$ and $Y \in \mathbb{N}$.

Step 1: Set $\mathcal{A}^{\text{fix}}_{\text{head}} := \{(i,j) \in \mathcal{A}_{\text{head}} : Y + L_{ij} \ > \ \pi_j - \pi_i \ > \ Y\}$.

Step 2: Set $\tilde{\mathcal{A}}_{\text{head}} := \{(i,j) \in \mathcal{A}_{\text{head}} : |\pi_j - \pi_i| \ \leq \ Y\}$.

Step 3: Define the network $\mathcal{N}'$ by

$$\begin{aligned} \mathcal{E}' &:= \mathcal{E} \\ \mathcal{A}'_{\text{train}} &:= \mathcal{A}_{\text{train}} \cup \mathcal{A}^{\text{fix}}_{\text{head}} \\ \mathcal{A}'_{\text{change}} &:= \mathcal{A}_{\text{change}} \\ \mathcal{A}'_{\text{head}} &:= \tilde{\mathcal{A}}_{\text{head}}. \end{aligned}$$

Output: The reduced event-activity network $\mathcal{N}'$.

It is important to notice that bounding the maximal delay of each event by Y as suggested in this section cuts off all solutions of (DM) where at least one event $i \in \mathcal{E}$ satisfies $x_i > \pi_i + Y$; this might even lead to an infeasible problem. So when solving (DM) on the reduced network $\mathcal{N}'$ provided by algorithm Fix-Headways, we might yield a solution that is not optimal for (DM), or we might not even get a solution at all. However, if we already start with an instance of (BDM) instead of (DM), then solving it on the reduced network $\mathcal{N}'$ is exactly the same as solving it on the original network $\mathcal{N}$ as we do not cut off any feasible solution of (BDM).

In Section 3.5, we show how much the reduced size affects the computation times in practice and how much the objective value differs from the optimal solution if we do not take into account all feasible solutions of (DM), but only those where the delay of each event is bounded (i.e. if we solve (DM) on the reduced event-activity network provided by algorithm Fix-Headways).

In the next section, we further analyze the headway constraints. Our analysis yields a result that allows us to reduce the size of the event-activity network; the resulting reduction technique can be combined with algorithm Fix-Headways to further reduce the size of the input instance.

3.3 Analyzing the Headway Constraints

Headway constraints increase the difficulty of the delay management problem as they can carry over a delay from a subsequent train even to a train which has been scheduled earlier. At a first glance, it seems that all of the headway activities can carry over a delay to a previous train. However, we prove that in an optimal solution, backward headways can never delay *punctual* events. This is precised next.

Definition 3.7. *Given the original timetable π, let $\mathcal{A}_\pi$ denote the set of all activities that are respected by π, i.e.*

$$\mathcal{A}_\pi := \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}^{\text{forw}}_{\text{head}}. \tag{3.4}$$

Using the definition of $\text{suc}(\cdot,\cdot)$ *from Section 2.1, we define the set of all successors of delays in the event-activity network $(\mathcal{E}, \mathcal{A}_\pi)$ as*

$$\mathcal{E}_{\text{mark}} := \mathcal{E}_{\text{mark}}(\mathcal{A}_\pi) := \left(\bigcup_{\substack{j \in \mathcal{E}: \\ d_j > 0}} \text{suc}(j, \mathcal{A}_\pi) \right) \cup \left(\bigcup_{\substack{a=(i,j) \in \mathcal{A}_{\text{train}}: \\ d_a > 0}} \text{suc}(j, \mathcal{A}_\pi) \right). \tag{3.5}$$

Note that source delayed events and endpoints of source delayed activities, as well as all of their successors in the event-activity network $(\mathcal{E}, \mathcal{A}_\pi)$, are included in this set. $\mathcal{E}_{\text{mark}}$ can be calculated in time $\mathcal{O}(|\mathcal{A}|)$.

In the following we show that $\mathcal{E}_{\text{mark}}$ contains all events to which (in an optimal solution) a delay can spread out in the worst case – although we do not consider backward headways in the set $\mathcal{A}_\pi$:

Theorem 3.8. *Let $w_i > 0 \ \forall i \in \mathcal{E}$ and let (x, z, g) be an optimal solution of (DM). Then, $x_i = \pi_i$ for all $i \notin \mathcal{E}_{\text{mark}}$.*

Proof. By contradiction. Let $i \notin \mathcal{E}_{\text{mark}}$ but $x_i > \pi_i$ where we assume that x_i is minimal among all such events. Since $i \notin \mathcal{E}_{\text{mark}}$, due to the definition of $\mathcal{E}_{\text{mark}}$, $d_i = 0$, and from the minimality of x_i, we conclude $x_k = \pi_k$ for all direct predecessors k with $(k, i) \in \mathcal{A}_\pi$. We show that reducing x_i to $\tilde{x}_i := \pi_i < x_i$ yields a feasible solution of (DM) with strictly better objective value. To this end, we show that $\tilde{x}$ fulfills constraints (2.2)-(2.5).

1. As $d_i = 0$, $\tilde{x}_i = \pi_i$ satisfies (2.2).
2. To show (2.3)-(2.5), we distinguish three types of activities: all outgoing activities, incoming activities except for backward headways, and incoming backward headways.

a) For all outgoing activities $a = (i, k) \in \mathcal{A}$, we obtain

$$x_k - \tilde{x}_i > x_k - x_i \geq L_a + d_a$$

where the last inequality holds since x is a feasible timetable. Consequently (2.3)-(2.5) hold for all outgoing activities $a = (i, k) \in \mathcal{A}$.

b) For all incoming activities $a = (k, i) \in \mathcal{A}_\pi$, we use $x_k = \pi_k$ and $d_a = 0$ to derive

$$\tilde{x}_i - x_k = \pi_i - \pi_k \geq L_a = L_a + d_a,$$

hence (2.3)-(2.5) hold for each incoming activity $a = (k, i) \in \mathcal{A}_\pi$.

c) As the last step, we have to check incoming backward headways, so let $(k, i) \in \mathcal{A}_{\text{head}} \setminus \mathcal{A}_\pi = \mathcal{A}_{\text{head}}^{\text{back}}$. If $g_{ki} = 1$, then – according to part 2a of this proof – the corresponding outgoing headway $a^{-1} = (i, k)$ with $g_{ik} = 0$ satisfies (2.5), so as a consequence of Corollary 3.2, a also satisfies (2.5).

Hence, assume $g_{ki} = 0$. We now show that changing the order of events k and i back to the order which they had in the original timetable π leads to a feasible timetable, enabling us to set $\tilde{x}_i = \pi_i$ without changing the time of event k. As $(k, i) \notin \mathcal{A}_\pi$, we have $\pi_k > \pi_i$, and as π is a feasible timetable, this implies $\pi_k - \pi_i \geq L_{ik}$. We define $\tilde{g}_{ki} := 1$ and $\tilde{g}_{ik} := 0$; then

$$x_k - \tilde{x}_i = x_k - \pi_i \geq \pi_k - \pi_i \geq L_{ik},$$

so (2.5) still holds for (i, k) if we swap the order of both events. Then, by the same argument as before, due to Corollary 3.2, (2.5) also holds for (k, i).

All in all, $(\tilde{x}, z, \tilde{g})$ is a feasible solution with strictly better objective value than (x, z, g) (as $w_i > 0$), a contradiction to the optimality of (x, z, g). □

If the weight of an event represents its importance, then the requirement $w_i > 0 \; \forall i \in \mathcal{E}$ in Theorem 3.8 is no severe restriction. If, however, the weight w_i of event i represents the number of passengers who end their journey with event i, then – as already mentioned in Section 2.3 – we need to set $w_i = 0$ for all departure events $i \in \mathcal{E}_{\text{dep}}$. In this case, delaying an event with weight 0 does not increase the objective value as long as it does not influence any event or connection with positive weight, so there might exist an optimal solution with $x_i > \pi_i$ for some event $i \notin \mathcal{E}_{\text{mark}}$. However, using the proof of Theorem 3.8, we can transform such a solution into a solution $(\tilde{x}, z, \tilde{g})$ with the same or even better objective value, but with $x_i = \pi_i$ for all $i \notin \mathcal{E}_{\text{mark}}$:

Theorem 3.9. *Let $w_i \geq 0 \; \forall i \in \mathcal{E}$, and let (DM) be feasible. Then there exists an optimal solution (x, z, g) of (DM) with $x_i = \pi_i$ for all $i \notin \mathcal{E}_{\text{mark}}$.*

Proof. Let (x, z, g) be an optimal solution of (DM) with $x_i > \pi_i$ for some events $i \in \mathcal{E}_{\text{mark}}$. By $\mathcal{E}_{\text{violated}} \subset \mathcal{E}_{\text{mark}}$, we denote all such events. Then, by the construction shown in the proof of Theorem 3.8, we can set $\tilde{x}_i := \pi_i$ for all $i \in \mathcal{E}_{\text{violated}}$ and obtain a feasible solution $(\tilde{x}, z, \tilde{g})$ of (DM). The objective value of this solution is not worse than the objective value of the initial optimal solution (x, z, g) since the set of maintained connections is the same as before (as we assumed $w_a > 0 \; \forall a \in \mathcal{A}_{\text{change}}$ in our model) and since no event takes place later than scheduled in the initial disposition timetable x. □

Theorem 3.8 implies that all events k with $x_k > \pi_k$ in an optimal solution are included in $\mathcal{E}_{\text{mark}}$. We can strengthen this result as follows:

Theorem 3.10. *Let $w_i > 0 \; \forall \; i \in \mathcal{E}$ and let (x, z, g) be an optimal solution of (DM). Let $x_k > \pi_k$ for some $k \in \mathcal{E}$. Then there exists a directed path p from some event $i \in \mathcal{E}$ to event k with either $d_i > 0$ or $d_a > 0$ for some $a = (j, i) \in \mathcal{A}_{\text{train}}$ such that*

$$p \subseteq (\mathcal{E}, \mathcal{A}') \text{ with } \mathcal{A}' := \mathcal{A}_\pi \setminus \{a \in \mathcal{A}_{\text{change}} : z_a = 1\}.$$

Proof. Let (x, z, g) be optimal for (DM) in the network $(\mathcal{E}, \mathcal{A})$. Consider the network $\mathcal{N}' = (\mathcal{E}, \mathcal{A}')$ in which the changing activities that have not been maintained in the optimal solution are deleted. The solution (x, z', g) (where z' is the restriction of z on $\{0,1\}^{|\mathcal{A}'|}$) is still optimal for the delay management problem in $\mathcal{N}'$. Since $x_k > \pi_k$, Theorem 3.8 yields that $k \in \mathcal{E}_{\text{mark}}(\mathcal{A}')$, i.e. the required path in $\mathcal{N}'$ exists according to the definition of $\mathcal{E}_{\text{mark}}$. □

Again, if $w_i \geq 0$ instead of $w_i > 0$, then (as shown in Theorem 3.9), for each optimal solution, there exists a corresponding solution with the same objective value and the property claimed in Theorem 3.10.

The results from Theorem 3.8 and Theorem 3.9 can be used to reduce the size of the IP formulation (DM). Formally, this is done by the following algorithm:

Algorithm 3.2: REDUCE

Input: The event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, a timetable π, and source delays $d \in \mathbb{N}^{|\mathcal{E}|+|\mathcal{A}_{\text{train}}|}$.

Step 1: Calculate $\mathcal{E}_{\text{mark}}$ according to (3.5), using the definition of $\mathcal{A}_\pi$ from (3.4).

Step 2: Compute $\mathcal{E}_{\text{reduced}} := \mathcal{E}_{\text{mark}} \cup \{i \in \mathcal{E} : \exists a = (i, j) \in \mathcal{A}_{\text{train}} \text{ with } d_a > 0\}$.

Step 3: Compute $\mathcal{A}_{\text{reduced}} := \{(i, j) \in \mathcal{A} : i, j \in \mathcal{E}_{\text{reduced}}\}$.

Output: The reduced event-activity network $\mathcal{N}_{\text{reduced}} = (\mathcal{E}_{\text{reduced}}, \mathcal{A}_{\text{reduced}})$.

$\mathcal{N}_{\text{reduced}}$ might be significantly smaller than $\mathcal{N}$, hence solving (DM) on the reduced event-activity network might be significantly faster than solving it on the original one. To obtain a solution for all events and activities after solving (DM) on the reduced network, we have to set $x_i = \pi_i$ for all events $i \in \mathcal{E} \setminus \mathcal{E}_{\text{reduced}}$, $z_a = 0$ for all changing activities $a \in \mathcal{A}_{\text{change}} \setminus \mathcal{A}_{\text{reduced}}$, $g_{ij} = 0$ for all forward headways $a = (i, j) \in \mathcal{A}^{\text{forw}}_{\text{head}} \setminus \mathcal{A}_{\text{reduced}}$ and $g_{ji} = 1$ for the corresponding backward headways.

Note that for calculating the constant M from Corollary 3.2, it is also sufficient to use the reduced network from above. This might lead to a much smaller value of M, yielding a better linear programming relaxation of the integer program and reducing rounding errors if the IP is solved by a commercial solver. We can also combine the results from Theorem 3.5, Theorem 3.6, and Theorem 3.8 (i.e. combine algorithms Fix-Headways and Reduce) to further reduce the size of the input instance. This is formalized in the following algorithm Fix-And-Reduce:

Algorithm 3.3: Fix-And-Reduce

Input: An instance of (DM) and $Y \in \mathbb{N}$.

Step 1: Run algorithm Fix-Headways and obtain $\mathcal{N}'$.

Step 2: Run algorithm Reduce on $\mathcal{N}'$ and obtain $\mathcal{N}_{\text{reduced}}$.

Step 3: Define $\mathcal{N}''$ by

$$\begin{aligned} \mathcal{E}'' &:= \mathcal{E}_{\text{reduced}} \subseteq \mathcal{E}' \\ \mathcal{A}''_{\text{train}} &:= \mathcal{A}'_{\text{train}} \cap \mathcal{A}_{\text{reduced}} \\ \mathcal{A}''_{\text{change}} &:= \mathcal{A}'_{\text{change}} \cap \mathcal{A}_{\text{reduced}} \\ \mathcal{A}''_{\text{head}} &:= \mathcal{A}'_{\text{head}} \cap \mathcal{A}_{\text{reduced}}. \end{aligned}$$

Output: The reduced event-activity network $\mathcal{N}''$.

Note that algorithm Fix-And-Reduce (like algorithms Fix-Headways and Reduce) requires only linear runtime of $\mathcal{O}(|\mathcal{A}| + |\mathcal{E}|)$.

In general, algorithm Fix-Headways (Step 1 of algorithm Fix-And-Reduce) should be applied before algorithm Reduce (Step 2 of algorithm Fix-And-Reduce): Algorithm Fix-Headways might delete some headways that otherwise would increase the set $\mathcal{E}_{\text{mark}}$; by deleting them, algorithm Reduce gets more effective. If, by contrast, Reduce is invoked before algorithm Fix-Headways, this has no effect on the effectiveness of Fix-Headways.

The reduced event-activity network $\mathcal{N}''$ computed by algorithm Fix-And-Reduce can be used to solve problem (BDM): First, set M as defined in (3.5) where it is sufficient

to use the reduced set $\mathcal{A}''_{\text{head}}$ instead of $\mathcal{A}_{\text{head}}$ and solve the corresponding instance of (BDM) on the reduced event-activity network $\mathcal{N}''$. If the problem is feasible, an optimal solution is also a *feasible* solution for problem (DM). In Section 3.5.4, we show that for a reasonable choice of Y, in our case study, the optimal solution of (BDM) is even *optimal* for (DM). To extend the solution (x, z, g) corresponding to the reduced event-activity network $\mathcal{N}''$ to a solution corresponding to the original event-activity network $\mathcal{N}$, we have to set $x_i = \pi_i \; \forall i \in \mathcal{E} \setminus \mathcal{E}''$, $z_a = 0 \; \forall a \in \mathcal{A}_{\text{change}} \setminus \mathcal{A}''_{\text{change}}$, $g_{ij} = 0 \; \forall (i,j) \in \mathcal{A}^{\text{forw}}_{\text{head}} \setminus \mathcal{A}''_{\text{head}}$, and $g_{ij} = 1 \; \forall (i,j) \in \mathcal{A}^{\text{back}}_{\text{head}} \setminus \mathcal{A}''_{\text{head}}$.

3.4 Never-Meet Property for Capacitated Delay Management

Using the results from Section 3.3, we can extend the never-meet property of uncapacitated delay management to the capacitated case. For results concerning the never-meet property of uncapacitated delay management, we refer to [Sch06].

Definition 3.11 ([Sch06])**.** *An instance of (UDM) has the* never-meet property *if*

- *for each source delay $d_j > 0$ with $j \in \mathcal{E}$ (or $d_a > 0$ with $a = (i,j) \in \mathcal{A}_{\text{train}}$), $\text{suc}(j, \mathcal{A})$ is an out-tree, and if*
- *for each pair of different source delays $d_j > 0$ (or $d_a > 0$, $a = (i,j)$) and $d_{\tilde{j}} > 0$ (or $d_{\tilde{a}} > 0$, $\tilde{a} = (\tilde{i}, \tilde{j})$), we have $\text{suc}(j, \mathcal{A}) \cap \text{suc}(\tilde{j}, \mathcal{A}) = \emptyset$.*

If the never meet-property is satisfied, (UDM) can be solved in time $O(|\mathcal{A}|)$ by dynamic programming, see [Sch06]. Moreover, the approximate objective function (2.1) is in this case the exact sum of all delays of all passengers when arriving at their final destinations (as long as the weights w_i and w_a are properly chosen as we already mentioned in Chapter 2). The reason for both results is that once a changing activity $a = (i,j)$ is not maintained, all its successors in $\text{suc}(i, \mathcal{A})$ are punctual and all its successive changing activities are maintained. For details and a study about the number of conflicts with the never-meet property for real-word instances, we refer to [Sch07].

If the never-meet property is not fulfilled, then we might over-estimate the sum of all delays in the objective of (UDM). To demonstrate this, consider the small event-activity network which we already have used in Section 3.3, see Figure 3.1 on page 26. Again we assume lower bounds $L_{(1,2)} = L_{(2,3)} = L_{(3,4)} = 1$, weights $w_1 = w_3 = 0$, $w_2 = w_{(2,3)} = 1$, and $w_4 = 3$ and a period length of $T = 10$. If both events 1 and 3 have a source delay, then $\text{suc}(1, \mathcal{A}) \cap \text{suc}(3, \mathcal{A}) = \{3, 4\} \neq \emptyset$, i.e. the never-meet property is not fulfilled. Hence, we assume source delays $d_1 = 5$ and $d_3 = 1$. In the optimal solution, $z_{(2,3)} = 1$,

i.e. the connection is not maintained. A passenger traveling from station A to station C hence misses the connection, leading to a penalty of the period length T added to the objective. However, the weight w_4, representing the number of passengers ending their trip with event 4, also includes this passenger (as we assumed all weights to be fixed; they are not updated dynamically), so when adding $w_4(x_4 - \pi_4)$ to the objective, this passenger is counted a second time.

Directly transferring Definition 3.11 to the delay management problem (DM) with headway constraints makes no sense: Due to the directed circles induced by the headway constraints, it will never be satisfied if a source delay can spread out to some headway activity (and hence also to its reverse). However, we can use the results from Theorem 3.8 and Theorem 3.10 and only use $\mathcal{A}_\pi$ (instead of $\mathcal{A}$) to obtain a reasonable definition of the never-meet property.

Definition 3.12. *An instance of (DM) has the* never-meet property *if*

- *for each source delay $d_j > 0$ with $j \in \mathcal{E}$ (or $d_a > 0$ with $a = (i,j) \in \mathcal{A}_{\text{train}}$),* $\text{suc}(j, \mathcal{A}_\pi)$ *is an out-tree, and if*
- *for each pair of different source delays $d_j > 0$ ($d_a > 0$, $a = (i,j)$) and $d_{\tilde{j}} > 0$ ($d_{\tilde{a}} > 0$, $\tilde{a} = (\tilde{i}, \tilde{j})$), we have* $\text{suc}(j, \mathcal{A}_\pi) \cap \text{suc}(\tilde{j}, \mathcal{A}_\pi) = \emptyset$.

Here, $\mathcal{A}_\pi$ is as defined in (3.4).

The crucial property is that in an optimal solution, all events following a connection that has not been maintained are on time. This is shown next.

Lemma 3.13. *Given an instance of (DM) with $w_i > 0$ $\forall i \in \mathcal{E}$ that satisfies the never-meet property and an optimal solution (x, z, g), assume that $z_a = 1$ for some $a = (i,j) \in \mathcal{A}_{\text{change}}$. Then, $x_k = \pi_k$ for all subsequent events $k \in \text{suc}(j, \mathcal{A}_\pi)$.*

Proof. By contradiction. Let (x, z, g) be an optimal solution with $z_a = 1$ for some $a = (i,j) \in \mathcal{A}_{\text{change}}$ and assume that there exists some event $k \in \text{suc}(j, \mathcal{A}_\pi)$ with $x_k > \pi_k$.

- On the one hand, according to Theorem 3.10, there exists a directed path p_1 from some event $\tilde{j}_1$ to k in $(\mathcal{E}, \mathcal{A}_\pi \setminus \{a\})$ with either $d_{\tilde{j}_1} > 0$ or $d_{\tilde{a}_1} > 0$ for some $\tilde{a}_1 = (\tilde{i}_1, \tilde{j}_1) \in \mathcal{A}_{\text{train}}$ that causes the delay of k.
- On the other hand, from $z_a = 1$, we conclude that i is delayed (otherwise it would have been better to maintain a as we assumed all weights w_a to be strictly positive). Hence, according to Theorem 3.8, $i \in \mathcal{E}_{\text{mark}}$, so there exists a directed path p_2 from some event $\tilde{j}_2$ to i in $(\mathcal{E}, \mathcal{A}_\pi \setminus \{a\})$ with either $d_{\tilde{j}_2} > 0$ or $d_{\tilde{a}_2} > 0$ for some $\tilde{a}_2 = (\tilde{i}_2, \tilde{j}_2) \in \mathcal{A}_{\text{train}}$ that causes the delay of i. As $k \in \text{suc}(j, \mathcal{A}_\pi)$, p_2 can be extended to a path p_3 from $\tilde{j}_2$ to k in $(\mathcal{E}, \mathcal{A}_\pi)$ that contains a.

Together, we obtain a contradiction to the never-meet property: either $\tilde{j}_1 = \tilde{j}_2$, then $\text{suc}(\tilde{j}_1, \mathcal{A}_\pi)$ is not an out-tree as we have two different paths p_1 (not containing a) and p_3 (containing a) from $\tilde{j}_1$ to k, or $\tilde{j}_1 \neq \tilde{j}_2$, then $\text{suc}(\tilde{j}_1, \mathcal{A}_\pi) \cap \text{suc}(\tilde{j}_2, \mathcal{A}_\pi) \supseteq \{k\} \neq \emptyset$. □

Note that due to Theorem 3.9, in the case that $w_i = 0$ is allowed, for each optimal solution (x, z, g) that does not fulfill Lemma 3.13 due to some event $i \notin \mathcal{E}_{\text{mark}}$ with $w_i = 0$ and $x_i > \pi_i$, we can construct a corresponding solution $(\tilde{x}, z, \tilde{g})$ that *does* fulfill Lemma 3.13.

Another important result is that if the never-meet property holds, then in each optimal solution, all subsequent connections following a dropped one are maintained:

Corollary 3.14. *Given an instance of (DM) that satisfies the never-meet property and an optimal solution (x, z, g), assume that $z_a = 1$ for some changing activity $a = (i, j) \in \mathcal{A}_{\text{change}}$. Then, for each subsequent changing activity $a = (k, l) \in \mathcal{A}_{\text{change}}$ with $k \in \text{suc}(j, \mathcal{A}_\pi)$, $z_a = 0$ holds.*

Proof. We distinguish two cases, depending on whether a weight of 0 is allowed for events or not.

1. If $w_i > 0 \ \forall i \in \mathcal{E}$, Lemma 3.13 yields that $x_k = \pi_k$. Hence it is feasible to set $z_a = 0$, and as we assumed that $w_a > 0$ for all $a \in \mathcal{A}_{\text{change}}$, it is strictly better than $z_a = 1$. Thus, in an optimal solution, $z_a = 0$.

2. If $w_i = 0$ is allowed, then (as a consequence of Theorem 3.9) there exists a corresponding solution $(\tilde{x}, z, \tilde{g})$ whose objective value it as least as good as the objective value of (x, z, g) and for which Lemma 3.13 holds. Hence, by the same argument as in the first part of this proof, the claim of Corollary 3.14 is true for $(\tilde{x}, z, \tilde{g})$. As the values of the binary variables z are the same in both solutions (x, z, g) and $(\tilde{x}, z, \tilde{g})$, Corollary 3.14 also holds for the initial solution (x, z, g). □

Lemma 3.13 and Corollary 3.14 provide the basic ingredients for most of the results based on the never-meet property. We mention the most important consequence here:

Theorem 3.15. *If the never-meet property holds, (DM) is equivalent to minimizing the sum of all delays of all passengers at their final destinations.*

Proof. The idea of the proof is similar to the proof for the uncapacitated case given in [Sch06]. If all connections on a passenger's path are maintained, this passenger arrives at his or her final destination with exactly the train he or she planned to use. As we add the delay of the corresponding arrival event, weighted with the number of passengers who end their journey with that event, to the objective (2.1), all passengers who do not miss a connection are counted correctly. It remains to show that a passenger

who misses a connection is not counted twice, i.e. that for such a passenger, a delay of exactly T is added to the objective (to model the fact that this passenger has to wait for a train of the same line in the next period as we prohibit re-routing of passengers).

To this end, assume that connection $a = (i, j) \in \mathcal{A}_{\text{change}}$ is not maintained, i.e. $z_a = 1$. Then Tw_a is added to the objective (2.1). Due to Lemma 3.13, $x_k = \pi_k$ for all subsequent events $k \in \text{suc}(j, \mathcal{A}_\pi)$, hence for all passengers who miss a, the delays of the trains they planned to use to reach their final destinations are $x_k - \pi_k = 0$ at all subsequent events $k \in \text{suc}(j, \mathcal{A}_\pi)$, thus no additional delay is added for these passengers at their final destinations. Due to Corollary 3.14, all subsequent connections of a missed one are maintained, so for no passenger (even if he or she planned to transfer several times), we add T more than once to the second part of the objective (2.1). □

The never-meet property for uncapacitated delay management is almost satisfied in many practical scenarios (for a study on how many conflicts with the never-meet property occur in a real-world railway setting, we refer to [Sch07]). However, in the capacitated case, conflicts with the never-meet property are more likely to appear as the event-activity network with headway activities is much more dense than without headways. Hence, in many cases, the objective (2.1) is only an upper bound on the sum of all delays of all passengers at their final destinations.

3.5 Numerical Results

In this section, we present numerical results from a case study with a real-world data set, based on a part of the German railway network, to show the effectiveness of the reduction techniques suggested in this chapter.

3.5.1 Setting

We start with a description of the setting of our numerical tests. The part of the German railway network which we consider in our case study is a part of the network of Deutsche Bahn AG in the southern part of Lower Saxony, namely the Harz region. It is depicted in Figure 3.2.

For the case study, we consider 30 pairs of directed passenger railway lines operated within the region as well as 9 pairs of long-distance lines crossing the region. The data set contains all 598 stations along these lines. In total, 92 trains are serving the lines, and for all tracks, including more than 30 single track segments, we take into account the headway constraints to ensure safe operations. The weights of the changing

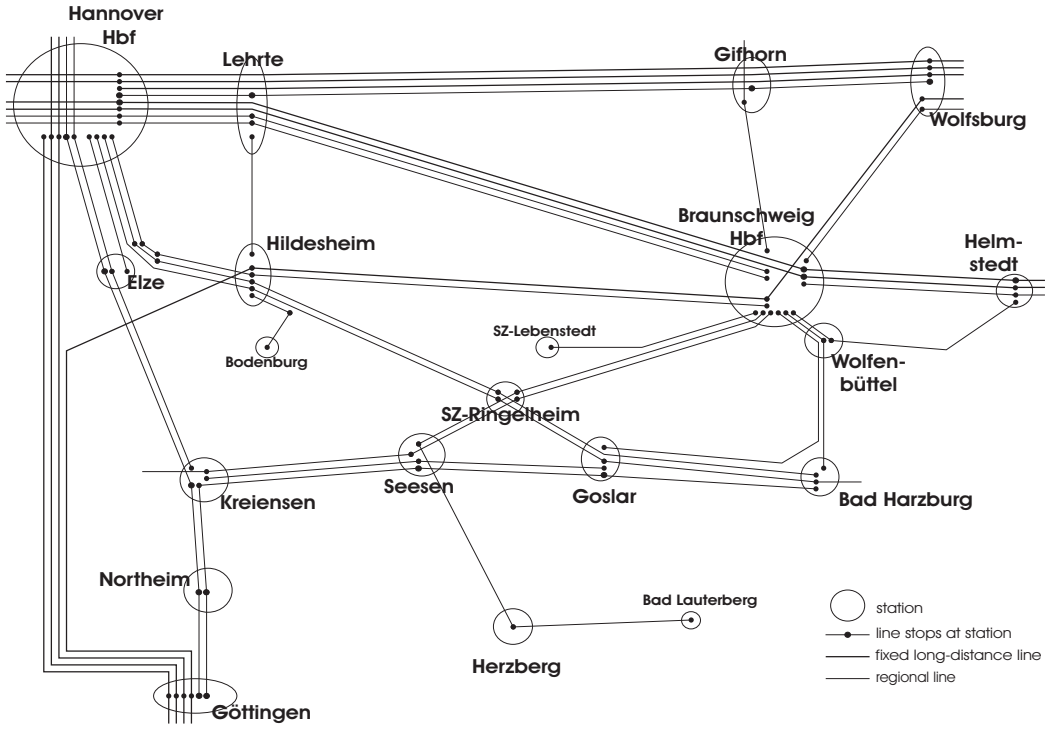

Figure 3.2: The part of the German railway network which we consider in our case study. The figure is taken from [LSS+10].

activities have been estimated by Deutsche Bahn AG using their traffic assignment model. We assume all lines to be operated with a common period of $T = 120$ minutes. In reality, there are only few lines which are operated hourly (which, for our case study, are split into two lines with a period of 120 minutes, operating alternately at one-hour intervals). The timetable of the long-distance lines crossing the region was fixed according to a timetable provided by Deutsche Bahn AG, while the timetable of the local trains has been computed for a joint research on delay resistant timetabling, minimizing the passengers' travel times (see [LSS+10]).

To evaluate the reduction techniques presented in this chapter, we take into account different observation periods, considering all events taking place in a fixed time interval of 2, 4, 6, 8, 10, or 12 hours and all activities connecting those events. There are two reasons for taking into account several different observation periods: First, a larger observation period results in a larger event-activity network which allows us to numerically evaluate the runtime of different solution procedures, depending on

the input size. Secondly, by allowing source delays to occur only during the first two hours of each observation period, we can analyze the relative error of different solution procedures on short-term, medium-term, and long-term disposition. The sizes of the corresponding event-activity networks are summarized in Table 3.3.

hours	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{A}_{\text{train}}\rvert$	$\lvert\mathcal{A}_{\text{change}}\rvert$	$\lvert\mathcal{A}_{\text{head}}\rvert$
2	5 320	5 219	151	1 204
4	10 637	10 529	380	4 804
6	15 953	15 843	609	10 800
8	21 269	21 157	838	19 192
10	26 004	25 890	1 036	28 612
12	28 336	28 220	1 142	33 466

Table 3.3: Size of the event-activity network, depending on the observation period.

For each of the observation periods, we generated 300 different delay scenarios; in each of them, ten randomly chosen driving activities within the first two hours of the observation period have been assigned a randomly chosen source delay between 3 and 15 minutes. For each delay scenario, we then used several different approaches to compute a solution of the corresponding instance of the delay management problem: namely solving the IP formulation based on the original event-activity network as well as solving the IP formulation based on each of the three different reduced event-activity networks obtained by applying Fix-Headways, Reduce, and Fix-And-Reduce. In the following, we compare the time needed for solving the IP based on the unreduced event-activity network with the time needed for solving the IP based on one of the reduced event-activity networks. The ratio of both times is a measure for the efficiency of the reduction techniques (note that we only take into account the time needed by the solver to solve the IP to optimality, not the time needed for setting up the IP or for applying one of the reduction techniques). The computations have been performed on five machines, each equipped with two dual core AMD Opteron 275 CPUs with 2.2 GHz and 12 GB RAM, using FICO Xpress 7.0 (using the default settings for the optimizer). In the following, we present the results.

3.5.2 Sizes of the Reduced Event-Activity Networks

First, we compare how much algorithms Fix-Headways, Reduce, and Fix-And-Reduce (which in fact combines Fix-Headways and Reduce) reduce the size of the input instance of (DM).

In Table 3.6 on page 48, we show how good the reduction results are for algorithm Fix-Headways (where we have chosen $Y = T$). Note that Fix-Headways only reduces the number of headway activities but does neither delete events nor driving, waiting, or changing activities. Furthermore, the reduction does not depend on the actual delay scenario, but only on the event-activity network and the upper bound Y on the maximal delay of an event – hence, for all delay scenarios belonging to a fixed observation period, the reduced event-activity network after running algorithm Fix-Headways is the same. As expected, the larger the observation period gets, the more headway activities are either fixed or even completely removed. As the period length T is equal to 120 minutes, for an observation period of two hours, no reduction is possible.

In Table 3.4, we show how much algorithm Reduce can reduce the size of the event-activity network. In contrast to algorithm Fix-Headways, it does not only reduce the number of headways, but also deletes some events and some driving, waiting, and changing activities and depends on the source delays. Reduce yields good reduction results especially for short observation periods, while the effectiveness of Fix-Headways (see Table 3.6) grows with the observation period. Combining both algorithms to algorithm Fix-And-Reduce yields a reduction technique that performs well for small *and* for large observation periods. This can bee seen in Table 3.5 where we summarize the reduction results for algorithm Fix-And-Reduce.

Depending on the network structure, choosing a smaller Y can increase the efficiency of algorithm Fix-Headways and algorithm Fix-And-Reduce. In this case, Fix-Headways can delete more headways, and the tighter upper bound on each x_i variable reduces the search space. Since in algorithm Fix-And-Reduce, Fix-Headways is run before Reduce, it also removes some headways that otherwise would increase the set $\mathcal{E}_{\text{mark}}$ – by deleting them, algorithm Reduce gets more effective. Note that this does not hold if $Y \geq T$; we explain this for the case of two trains using the same track into the same direction. A headway (i, j) can be deleted only if $\pi_j - \pi_i \geq Y + L_{ij}$. If this is the case, then (due to the periodicity) there exist other events $i_1, \ldots, i_k$ with $\pi_{i_1} = \pi_i + T, \ldots, \pi_{i_k} = \pi_{i_{k-1}} + T \leq \pi_j - T$ which are connected by headway activities $(i, i_1), \ldots, (i_{k-1}, i_k), (i_k, j)$. As $Y \geq T$, those headway activities are not deleted by algorithm Fix-Headways. Hence when Reduce is running, due to those headways, if $i \in \mathcal{E}_{\text{mark}}$, j gets marked, too, as $j \in \text{suc}(i, \mathcal{A}_\pi)$.

As with a smaller value of Y, the problem got infeasible in few cases, we only present results for $Y = T$.

	reduced size of $\mathcal{E}$			reduced size of $\mathcal{A}_{\text{change}}$			reduced size of $\mathcal{A}_{\text{head}}$		
hours	max	avg	min	max	avg	min	max	avg	min
2	29.10%	13.94%	4.08%	29.80%	9.06%	0.00%	39.70%	13.32%	0.00%
4	53.18%	34.08%	17.01%	68.95%	36.32%	5.53%	65.90%	34.70%	5.38%
6	62.73%	49.43%	34.42%	74.55%	55.46%	27.75%	77.33%	49.32%	17.15%
8	71.84%	60.76%	44.31%	81.03%	67.32%	49.88%	79.77%	60.19%	31.11%
10	77.58%	67.33%	55.67%	86.29%	73.59%	60.23%	85.58%	66.77%	42.60%
12	78.48%	70.34%	59.84%	85.64%	76.25%	63.31%	83.88%	69.28%	49.85%

Table 3.4: Relative size of the event-activity network (compared to the unreduced event-activity network) after running algorithm Reduce.

	reduced size of $\mathcal{E}$			reduced size of $\mathcal{A}_{\text{change}}$			reduced size of $\mathcal{A}_{\text{head}}$		
hours	max	avg	min	max	avg	min	max	avg	min
2	29.10%	13.94%	4.08%	29.80%	9.06%	0.00%	39.70%	13.32%	0.00%
4	53.18%	34.08%	17.01%	68.95%	36.32%	5.53%	54.33%	31.49%	5.79%
6	62.73%	49.43%	34.42%	74.55%	55.46%	27.75%	46.50%	34.13%	14.65%
8	71.84%	60.76%	44.31%	81.03%	67.32%	49.88%	37.91%	31.86%	20.38%
10	77.58%	67.33%	55.67%	86.29%	73.59%	60.23%	33.29%	28.78%	22.07%
12	78.48%	70.34%	59.84%	85.64%	76.25%	63.31%	29.95%	26.76%	21.91%

Table 3.5: Relative size of the event-activity network (compared to the unreduced event-activity network) after running algorithm Fix-And-Reduce.

hours	reduced size of $\mathcal{A}_{\text{head}}$	deleted headways
2	100.00%	0
4	74.94%	1 204
6	55.52%	4 804
8	43.73%	10 800
10	36.64%	18 128
12	33.43%	22 278

Table 3.6: Relative size of $\mathcal{A}_{\text{head}}$ (compared to the unreduced event-activity network) and number of deleted headways after running algorithm Fix-Headways.

3.5.3 Computation Times on Reduced Event-Activity Networks

Now we show how much the three different reduction techniques suggested in this chapter influence the computation time for solving the delay management problem. The results are given in Table 3.7 where we summarize the maximal, average, and minimal computation times for solving the IP on the reduced event-activity network, compared to the computation time needed for solving the IP on the unreduced event-activity network. In Figure 3.8 and Figure 3.9, we also depict the average computation times. The results are consistent with the results on how much the reduction techniques reduce the size of the event-activity network (see Table 3.6, Table 3.4, and Table 3.5): algorithm Reduce works good for short observation periods, while algorithm Fix-Headways performs better than Reduce only for large observation periods. Combining both methods in algorithm Fix-And-Reduce yields a significant reduction of the computation time for all observation periods.

Note that for an observation period of 2 hours, algorithm Fix-Headways does *not* reduce the size of the event-activity network as we have shown in Table 3.6, while it *does* reduce the computation time to about one third in average, see Table 3.7 and Figure 3.9. This is due to the fact that we explicitly add an upper bound on the x variables in problem (BDM) (which cuts off some feasible solutions and reduces the search space for the solver) and due to the fact that we can use a significantly smaller value for M in (BDM) than in (DM), yielding a stronger linear programming relaxation.

Despite of the fact that all three algorithms yield a decreased average computation time, especially for larger observation periods, in very few cases, the computation time even *grows* after applying a reduction technique. We suppose this is caused by the fact that the solver does not work exactly, but numerically within a given accuracy: Applying a reduction technique cuts off solutions that are feasible (but, in general, not

optimal, except for some cases where Fix-Headways and Fix-And-Reduce cut off optimal solutions with a too large delay of some single event). If the objective value of such a feasible, non-optimal solution is close to the objective value of an optimal one (i.e. "optimal" within the numerical accuracy of the solver), then – depending on the branching strategy in the branch and bound process – it might happen that the optimizer finds this solution rather quickly when solving the initial unreduced problem. However, as this solution is not feasible for the reduced problem, the optimizer has to put more effort in finding an optimal solution for the reduced problem, yielding an increased computation time. A similar effect can be caused by rounding errors as already demonstrated in the beginning of Section 3.1: for a large value of M (as in problem (DM)), an infeasible solution might be accepted as feasible by the optimizer, due to rounding errors and due to the limited numerical accuracy. For a smaller value of M (after applying the reduction technique), this solution might not be accepted as feasible by the solver – hence, again the solver has to put more effort in finding a different solution.

	Fix-Headways			Reduce		
hours	max	avg	min	max	avg	min
2	103.23%	37.30%	8.55%	58.18%	6.81%	0.01%
4	83.36%	24.32%	3.88%	85.74%	16.16%	0.62%
6	124.00%	25.88%	2.23%	107.41%	15.10%	2.01%
8	186.01%	33.52%	0.93%	192.42%	46.18%	2.70%
10	398.79%	37.42%	1.64%	603.65%	66.00%	4.63%
12	378.14%	34.14%	0.76%	778.54%	71.61%	9.53%

	Fix-And-Reduce		
hours	max	avg	min
2	34.71%	4.62%	0.01%
4	45.22%	8.49%	0.37%
6	40.94%	3.93%	0.87%
8	86.01%	14.80%	0.56%
10	187.07%	20.39%	1.07%
12	158.98%	20.02%	0.62%

Table 3.7: Computation time for solving the IP on the reduced event-activity network gained by applying different reduction techniques (in % of the time needed for solving the IP on the unreduced event-activity network).

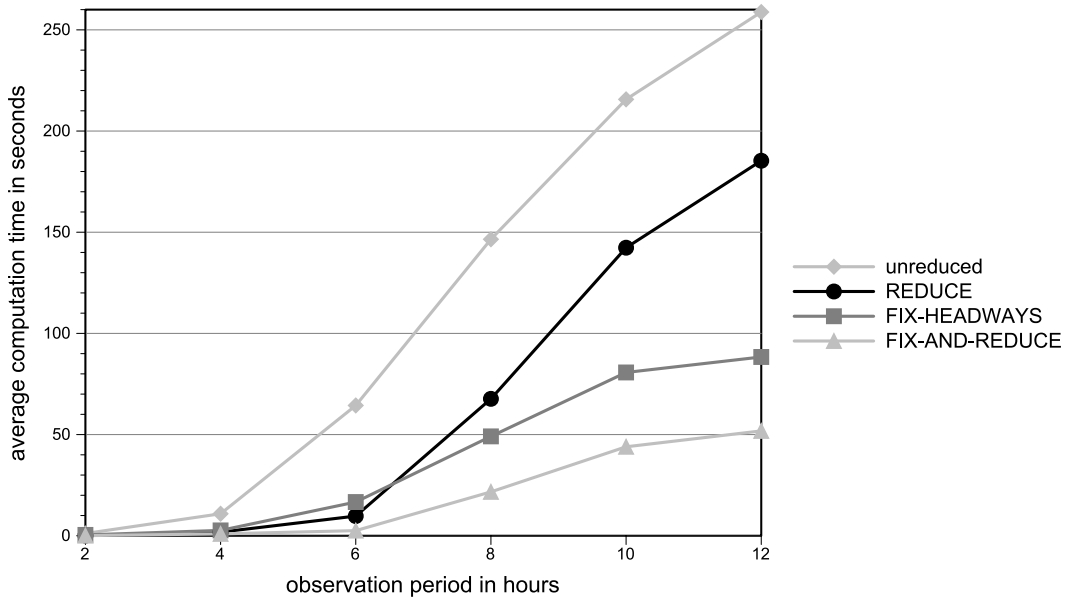

Figure 3.8: Average computation times for solving the IP on the unreduced network and on the reduced network gained by one of the reduction techniques, depending on the size of the observation period.

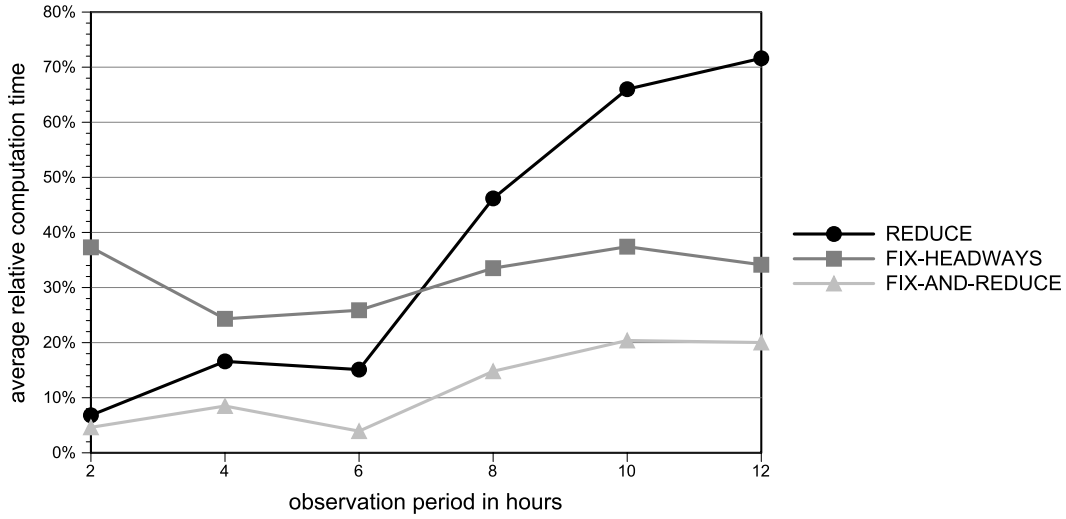

Figure 3.9: Average relative computation times for solving the IP after applying one of the reduction techniques (in % of the computation time on the unreduced network), depending on the size of the observation period.

3.5.4 Relative Error of Algorithm Fix-And-Reduce

Finally, we present numerical results for the relative error of the approach of solving the corresponding instance of (BDM) instead of solving the original instance of (DM). As mentioned in Section 3.2, by bounding the maximal delay of each event, we might cut off optimal solutions, so applying algorithm Fix-Headways might lead to a degradation of the objective value. However, during our case study (where we have chosen $Y = T$), it turned out that not only in all tested scenarios, the corresponding instance of (BDM) has been feasible, but also that *within the numerical accuracy of the solver*, the objective value did not get worse. The results are summarized in Table 3.10 where we present the maximal relative error over all scenarios, depending on the observation period.

hours	maximal relative error
2	0.00000
4	0.00000
6	0.00000
8	0.00000
10	0.00000
12	0.00001

Table 3.10: Maximal relative error of algorithm Fix-And-Reduce.

3.6 Summary

In Section 3.1, we proved that the constant M in the IP formulation (DM) indeed can be chosen finitely beforehand. In Section 3.2, we introduced a modification of problem (DM) for which we were able to proof a significantly smaller upper bound on this constant. Our analysis of the headway constraints in Section 3.3 shows that in an optimal solution, backward headways never carry over a delay to a *punctual* event (although they might cause additional delays on events that already are delayed). This result allowed us to extend the never-meet property and some of its consequences from uncapacitated delay management to the capacitated case in Section 3.4. However, in contrast to uncapacitated delay management, the never-meet property is only satisfied in few cases (compared to uncapacitated delay management, the event-activity network with headway activities is much more dense, hence a delay might spread out to a larger part of the event-activity network).

Based on the theoretical results from Sections 3.1-3.3, we suggested three different reduction techniques to reduce the size of an input instance of (DM):

- Algorithm REDUCE is based on the results from Section 3.3; it reduces the event-activity network by deleting all events (and all corresponding activities) to which no source delay can spread out in an optimal solution. If the weights of all events are strictly positive, it does not cut off any optimal solution. If some weights are zero and if the optimal solution is not unique, it might cut off some optimal solutions, but at least one optimal solution keeps feasible.

- Algorithm FIX-HEADWAYS is based on the results from Section 3.2. By adding an upper bound on the delay of each single event to the integer program, many headway constraints are fulfilled "automatically" in each feasible solution and hence can be deleted. Deleting them does not cut off any feasible solution of problem (BDM). Algorithm FIX-HEADWAYS can also be used to solve problem (DM) since according to Lemma 3.3, (DM) is a relaxation of (BDM) and a feasible solution of (BDM) hence is also feasible for (DM). However, in this case, FIX-HEADWAYS cuts of feasible solutions of (DM) and might even yield an infeasible problem (if for example $d_i > Y$ for some event $i \in \mathcal{E}$). Hence, the parameter Y has to be chosen carefully. Furthermore, an optimal solution of (BDM) does not have to be optimal for the relaxed problem (DM). However, for the data set of our case study and for $Y = T$, all optimal solutions of (BDM) turned out to be optimal for (DM), too.

- Algorithm FIX-AND-REDUCE is a combination of the other approaches: First, the maximal delay of each single event is bounded and algorithm FIX-HEADWAYS is applied to delete the headway activities that are respected "automatically", then algorithm REDUCE is applied to the resulting event-activity network to delete all events (and all corresponding activities) to which no delay can spread out in an optimal solution. Depending on the size of Y and the structure of the event-activity network, running algorithm FIX-HEADWAYS might delete some headways that otherwise would increase the set $\mathcal{E}_{\text{mark}}$ calculated by algorithm REDUCE; this might improve the efficiency of algorithm REDUCE as more events might be deleted.

Apart from the reduction of the size of an input instance, there is another reason for applying one of the reduction techniques before solving the IP: it is sufficient to compute the constant M for problem (DM) as defined in Theorem 3.1 and Corollary 3.2 and the constant M for problem (BDM) as defined in Theorem 3.5 for the reduced event-activity networks provided by algorithms FIX-HEADWAYS, REDUCE, or FIX-AND-REDUCE instead of the original event-activity network – this might lead to a significantly smaller value of M and hence to a better linear programming relaxation.

In addition, this might reduce the impact of rounding errors occurring when solving the IP formulation with a commercial solver as explained in Section 3.1.

After analyzing the model and the integer programming formulation, we numerically evaluated the reduction techniques suggested in this chapter. The results of our case study show that algorithm REDUCE yields good reduction results (concerning both the size of the input instance and the computation time) for small observation periods while algorithm FIX-HEADWAYS is superior for large observation periods. However, due to rounding errors and the limited numerical accuracy of the solver, especially for larger observation periods, there are few instances on which those reduction techniques increase the computation time significantly (while *in average*, both algorithms still reduce the computation time noticeably).

Combining both algorithms in algorithm FIX-AND-REDUCE yields a reduction technique that is well suited for small as well as for long observation periods. Furthermore, the number of cases in which its computation time is larger than the computation time on the unreduced event-activity network is significantly smaller than for FIX-HEADWAYS and REDUCE; the same holds for the maximal increase of the computation time.

Finally, within the numerical accuracy of the solver, the objective value did not get worse, hence $Y = T$ is a reasonable choice for the scenarios in our case study.

Chapter 4

Solution Approaches

In this chapter, we suggest different solution procedures for solving the delay management problem. One possibility of course is to solve the integer programming formulation (2.1)-(2.9) (either directly or after reducing the size of the input instance by one of the preprocessing procedures suggested in Chapter 3) by invoking a solver like CPLEX, FICO Xpress, or GLPK. However, it might take a very long time to do so on large-scale real-world instances as the problem turns out to be quite hard to solve, even if either $\mathcal{A}_{\text{head}} = \emptyset$ or $\mathcal{A}_{\text{change}} = \emptyset$:

Theorem 4.1 ([GJPS05])**.** *The uncapacitated delay management problem with the path-based objective* (2.11) *is NP-hard, even for very special input instances.*

Theorem 4.2 ([BHLS08])**.** *The online version of the uncapacitated delay management problem with a path-based objective (where delays occur one after another and are not known in advance) is PSPACE-hard.*

Theorem 4.3 ([CS07])**.** *Problem (Re-Sched) is NP-hard.*

Corollary 4.4. *Problem (DM) is NP-hard.*

Theorem 4.1 can be shown by reducing the *Maximum Independent Set* problem to a very simple uncapacitated delay management problem. By the reduction of the *Quantified Boolean Formula* problem to a simplified online version of the uncapacitated delay management problem, the PSPACE-hardness claimed in Theorem 4.2 can be proven. Finally, the proof of Theorem 4.3 works by reducing the *Job Shop Scheduling* problem to a special case of the capacitated delay management problem. Corollary 4.4 is a direct consequence of Theorem 4.3.

These results give rise to the assumption that no efficient algorithm for solving the delay management problem to optimality exists, provided that $P \neq NP$ holds. Although it

is possible to solve smaller instances to optimality in a reasonable amount of time, fast solution procedures for large-scale real-world instances are needed, especially as delay management is a problem that in practice has to be solved in realtime. To this end, we present two classes of heuristic solution approaches.

In Section 4.1, we introduce priority-based heuristics which heuristically fix the priority decisions and solve the remaining problem with fixed headway activities afterwards. Parts of that section are based on the results of a joint work and have been published in [SS08], see also [SS10].

In Section 4.2, we present the class of relax & repair heuristics. They are characterized by first relaxing the priority decisions, then solving the remaining problem and applying a repair strategy afterwards. That section is based on our presentation of those algorithms in [Sch09a].

Finally, in Section 4.3, we evaluate both classes of heuristics numerically, based on the same data set as in Section 3.5. We compare the runtime and the relative error of the heuristics and suggest a combination of different heuristics which combines their advantages and performs quite well in our case study.

4.1 Priority-Based Heuristics

The idea of priority-based heuristics is to fix the order of trains heuristically and to solve the remaining uncapacitated delay management problem afterwards. This approach is due to the observation that priority decisions make the problem "harder" to solve than wait/depart decisions as the number of priority decisions might be much larger than the number of wait/depart decisions (see Table 3.3 on page 45). In the following, we present four different heuristics from this class and a theoretical worst-case error analysis of these heuristics.

The first approach is to fix the g_{ij} variables according to the original timetable π, i.e. to keep the order of trains fixed, compared to the original timetable. Mathematically, $g_{ij} = 0$ if $\pi_i < \pi_j$ and $g_{ij} = 1$ if $\pi_i > \pi_j$. Then, an optimal solution of the remaining uncapacitated delay management problem is computed. Formally:

Algorithm 4.1: First scheduled, first served (FSFS)

Step 1: Set $\mathcal{A}_{\text{fix}} := \mathcal{A}^{\text{forw}}_{\text{head}}$.

Step 2: Compute an optimal solution of (UDM($\mathcal{A}_{\text{fix}}$)).

The next heuristic pursues a different strategy for fixing the priority decisions: First, all capacity constraints are neglected and the resulting uncapacitated delay management problem is solved to optimality. In the second step, the resulting disposition timetable x is used to fix the order of trains. Mathematically, $g_{ij} = 0$ if $x_i < x_j$ and $g_{ij} = 1$ if $x_i > x_j$. Then, again an optimal solution of the remaining uncapacitated delay management problem (with fixed headways) is computed. Formally:

Algorithm 4.2: First rescheduled, first served (FRFS)

Step 1: Solve the corresponding problem (UDM) and obtain a solution (x, z).

Step 2: Set $\mathcal{A}_{\text{fix}} := \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\}$.

Step 3: Compute an optimal solution of (UDM($\mathcal{A}_{\text{fix}}$)).

The running time of both heuristics further can be reduced by not only fixing the priority decisions heuristically, but also the wait/depart decisions. Modifying FRFS in this way yields the following solution approach for the delay management problem where (UDM) only has to be solved once, not twice as in FRFS:

Algorithm 4.3: FRFS with early connection fixing (FRFS-FIX)

Step 1: Solve the corresponding problem (UDM) and obtain a solution (x, z).

Step 2: Set $\mathcal{A}_{\text{fix}} := \{a \in \mathcal{A}_{\text{change}} : z_a = 0\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\}$.

Step 3: Compute the solution of (PP($\mathcal{A}_{\text{fix}}$)).

Also FSFS can be modified to fix the wait/depart decisions heuristically, yielding a polynomial-time algorithm. The idea is to maintain the "most important" connections and do not care about the less important ones.

Algorithm 4.4: FSFS with weight-based connection fixing (FSFS-FIX)

Step 1: Maintain the "most important" connections:

a) Sort $\mathcal{A}_{\text{change}}$ in descending order according to the weights w_a.

b) Set $z_a = 0$ for the first k% of the connections.

Step 2: Set $\mathcal{A}_{\text{fix}} := \{a \in \mathcal{A}_{\text{change}} : z_a = 0\} \cup \mathcal{A}^{\text{forw}}_{\text{head}}$.

Step 3: Compute the solution of (PP($\mathcal{A}_{\text{fix}}$)).

Since the first step in both FRFS and FRFS-FIX is to solve (UDM), we obtain from these heuristics not only an approximation of the optimal solution, but also a lower bound on its objective value, hence their maximal absolute errors can be bounded a posteriori:

Lemma 4.5. *Denote by* F^{FRFS} *and* $F^{\text{FRFS-FIX}}$ *the objective values of* FRFS *and* FRFS-FIX *and by* F^{DM} *the objective value of the optimal solution. Then:*

$$F^{\text{FRFS}} - F^{\text{DM}} \leq F^{\text{FRFS}} - F^{\text{UDM}},$$
$$F^{\text{FRFS-FIX}} - F^{\text{DM}} \leq F^{\text{FRFS-FIX}} - F^{\text{UDM}}.$$

Proof. As (UDM) is a relaxation of (DM) (see Lemma 2.1), we have $F^{\text{UDM}} \leq F^{\text{DM}}$. □

Similar error bounds of course also hold for FSFS and FSFS-FIX. However, as solving the corresponding instance of (UDM) is not part of both heuristics, computing these error bounds requires additional computations while FRFS and FRFS-FIX provide them "for free".

Comparing FRFS-FIX with FRFS as well as FSFS-FIX with FSFS yields the following results:

Lemma 4.6. *Denote by* F^{FRFS} *and* $F^{\text{FRFS-FIX}}$ *the objective values of* FRFS *and* FRFS-FIX *and by* F^{DM} *the objective value of the optimal solution. Then*

$$F^{\text{DM}} \leq F^{\text{FRFS}} \leq F^{\text{FRFS-FIX}}.$$

Proof. As FRFS always yields a feasible solution of (DM), the first inequality holds. As FRFS and FRFS-FIX fix the priority decisions in the same way, but FRFS-FIX uses a fixed rule for fixing the wait/depart decisions (which is only one of the options that FRFS has when computing optimal wait/depart decisions), the second inequality also holds. □

Lemma 4.7. *Denote by* F^{FSFS} *and* $F^{\text{FSFS-FIX}}$ *the objective values of the heuristics* FSFS *and* FSFS-FIX *and by* F^{DM} *the objective value of the optimal solution. Then*

$$F^{\text{DM}} \leq F^{\text{FSFS}} \leq F^{\text{FSFS-FIX}}.$$

Proof. Similar to the proof of Lemma 4.6. □

All those bounds of course are a posteriori bounds. We now present some a priori bounds for the heuristics suggested so far. First, we show that fixing the priority decisions according to the original timetable or according to an optimal solution of the corresponding instance of the uncapacitated delay management problem might lead

to an arbitrarily large relative error. Hence, the absolute and relative errors of the heuristics which we have presented cannot be bounded in general. However, we are able to bound the relative error of FRFS-FIX and FRFS, using the input data of the special instance (see Theorem 4.11 and Corollary 4.12).

Theorem 4.8. *Let* HEU *be a heuristic that solves (DM), fixing thethe order of trains as it is in the original timetable, and let* F^{HEU} *be its objective value. Then for each* $k \in \mathbb{N}$*, there exists an instance of (DM) with relative error*

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} > k.$$

Proof. Let $k \in \mathbb{N}$. Assume that we have two stations A and B and $k+2$ trains $h_0, h_1, \ldots, h_{k+1}$. All trains drive from station A to station B. By (h_t, A, dep) and (h_t, B, arr), we denote the departure of train $h_t \in \{h_0, \ldots, h_{k+1}\}$ from station A and its arrival at station B, respectively. $\pi(h_t, A, \mathrm{dep})$ and $\pi(h_t, B, \mathrm{arr})$ denote the originally scheduled times of those events. In the original timetable, the trains in our instance leave station A in the order $h_0, h_1, \ldots, h_{k+1}$ at the times $\pi(h_i, A, \mathrm{dep}) = i$ and arrive at station B at the times $\pi(h_i, B, \mathrm{arr}) = i + 1$, $i \in \{0, \ldots, k+1\}$. For each $i \in \{0, \ldots, k\}$, the departure of train h_i and the departure of train h_{i+1} are connected by a pair of headway activities (to keep it simple, we neglect headways between trains h_i and h_{i+2}, h_{i+3}, etc.). The weights of all departure events (h_t, A, dep) are set to 0, the weights of all arrival events (h_i, B, arr) are set to 1. The lower bounds of all activities are set to 1. The resulting event-activity network is shown in Figure 4.1.

Now, assume that (h_0, A, dep) is delayed by $d \geq k + 2$. One feasible solution is that trains $h_1, \ldots, h_{k+1}$ leave and arrive on time while train h_0 has a delay of d, so for

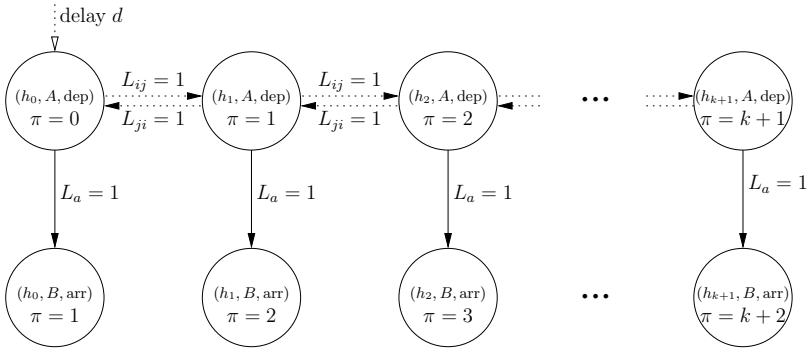

Figure 4.1: Worst-case example for Theorem 4.8. Solid arrows represent driving activities, dotted arrows represent headway activities.

the optimal solution, $F^{\mathrm{DM}} \leq d$. If we solve the problem by a heuristic that fixes the order of trains as it is in the original timetable, all trains get a delay of at least d, so $F^{\mathrm{HEU}} \geq (k+2) \cdot d$, hence

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} \;\geq\; \frac{(k+2) \cdot d - d}{d} \;=\; k+1 \;>\; k. \qquad \square$$

Similarly, one can show the following result concerning FSFS and FSFS-FIX:

Theorem 4.9. *Let* HEU *be a heuristic that solves (DM), fixing the headway activities according to the optimal solution of the corresponding problem (UDM). Then for each $k \in \mathbb{N}$, there exists an instance of (DM) with relative error*

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} \;>\; k.$$

Proof. Let $k \in \mathbb{N}$. Assume that we have two stations A and B and two trains g and h. Both trains drive from station A to station B. In the original timetable, the trains leave station A in the order g, h at the times $\pi(g, A, \mathrm{dep}) = 0$ and $\pi(h, A, \mathrm{dep}) = 1$ and arrive at station B at the times $\pi(g, B, \mathrm{arr}) = 1$ and $\pi(h, B, \mathrm{arr}) = 2$. The departures of both trains are connected by a pair of disjunctive headway activities with lower bounds 1 (headway activity from (g, A, dep) to (h, A, dep)) and $4 \cdot (k+1)$, respectively. All other lower bounds are set to 1. The weights of all departure events are set to 0, the weights of all arrival events to 1. The resulting event-activity network is shown in Figure 4.2.

Now, assume that (g, A, dep) is delayed by 2. In the optimal solution, both trains get a delay of 2, so $F^{\mathrm{DM}} = 4$. In the optimal solution of the relaxation without capacity constraints, train h departs and arrives on time and train g has a delay of 2. If the heuristic fixes the headway activities in this way, it has to respect the headway activity

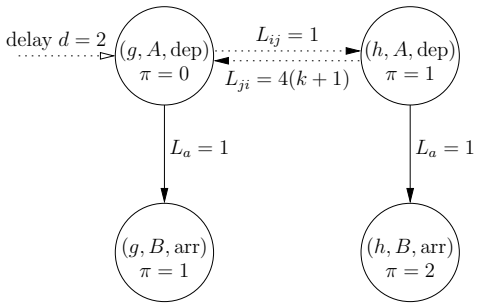

Figure 4.2: Event-activity network for the proof of Theorem 4.9. Solid arrows represent driving activities, dotted arrows represent headway activities.

with $L_{ij} = 4 \cdot (k+1)$ in the next step, so train h is on time while train g has a delay of at least $1 + 4 \cdot (k+1)$, hence

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} \;\geq\; \frac{1 + 4 \cdot (k+1) - 4}{4} \;=\; \frac{1 + 4k}{4} \;>\; k. \qquad \square$$

However – as we show in the numerical results at the end of this section – the heuristics do not behave as bad as one might think regarding these results. Moreover, using the input data of the actual instance, we are able to derive an upper bound on the relative errors of Frfs-Fix and Frfs. To this end, we need the following lemma:

Lemma 4.10. *Assume that for an instance of (DM), all wait/depart decisions and all priority decisions are already made, and let* $\mathcal{A}^{\mathrm{fix}}_{\mathrm{change}} \subseteq \mathcal{A}_{\mathrm{change}}$ *and* $\mathcal{A}^{\mathrm{fix}}_{\mathrm{head}} \subset \mathcal{A}_{\mathrm{head}}$ *be the sets reflecting those decisions. Furthermore, let* $\mathcal{A}_{\mathrm{fix}} := \mathcal{A}^{\mathrm{fix}}_{\mathrm{change}} \cup \mathcal{A}^{\mathrm{fix}}_{\mathrm{head}}$*, and let*

- x^{cap} *be an optimal solution of the "capacitated" problem (PP(*$\mathcal{A}_{\mathrm{train}} \cup \mathcal{A}_{\mathrm{fix}}$*)) and*
- x^{relax} *be an optimal solution of the "relaxed" problem (PP(*$\mathcal{A}_{\mathrm{train}} \cup \mathcal{A}^{\mathrm{fix}}_{\mathrm{change}}$*)) where* $\mathcal{A}^{\mathrm{fix}}_{\mathrm{head}}$ *is neglected.*

Furthermore, denote by

$$\mathcal{A}^{\text{fix-fw}}_{\mathrm{head}} := \mathcal{A}^{\mathrm{fix}}_{\mathrm{head}} \cap \mathcal{A}^{\mathrm{forw}}_{\mathrm{head}}$$
$$\mathcal{A}^{\text{fix-bw}}_{\mathrm{head}} := \mathcal{A}^{\mathrm{fix}}_{\mathrm{head}} \cap \mathcal{A}^{\mathrm{back}}_{\mathrm{head}}$$

the fixed forward headways and the fixed backward headways. Then, for each event $i \in \mathcal{E}$*,*

$$x_i^{\mathrm{cap}} \;\leq\; x_i^{\mathrm{relax}} + \sum_{\substack{k \in \mathrm{pre}(i):\\ \exists (k,l) \in \mathcal{A}^{\text{fix-fw}}_{\mathrm{head}}}} (x_k^{\mathrm{relax}} - \pi_k) + \sum_{\substack{k \in \mathrm{pre}(i):\\ \exists (k,l) \in \mathcal{A}^{\text{fix-bw}}_{\mathrm{head}}}} (x_k^{\mathrm{relax}} + \max_{(k,l) \in \mathcal{A}^{\text{fix-bw}}_{\mathrm{head}}} L_{kl}) \quad (4.1)$$

where $\mathrm{pre}(i) = \mathrm{pre}(i, \mathcal{A}_{\mathrm{train}} \cup \mathcal{A}_{\mathrm{fix}})$ *denotes the predecessors of event* i *in the event-activity network* $(\mathcal{E}, \mathcal{A}_{\mathrm{train}} \cup \mathcal{A}_{\mathrm{fix}})$ *as defined in Section 2.1.*

Proof. Given the sets $\mathcal{A}^{\mathrm{fix}}_{\mathrm{change}}$ and $\mathcal{A}^{\mathrm{fix}}_{\mathrm{head}}$, we can compute the solutions x^{cap} and x^{relax} of problems (PP($\mathcal{A}_{\mathrm{train}} \cup \mathcal{A}_{\mathrm{fix}}$)) and (PP($\mathcal{A}_{\mathrm{train}} \cup \mathcal{A}^{\mathrm{fix}}_{\mathrm{change}}$)) by applying the Critical Path Method, see Algorithm 2.1 on page 21. Cpm also computes a topological sorting of the event-activity network which we denote by $\prec$ and which we use to prove the claim by induction.

Basis: For the first event i_1, $x^{\mathrm{cap}}_{i_1} = \pi_{i_1} + d_{i_1} = x^{\mathrm{relax}}_{i_1}$, so (4.1) holds.

Inductive step: Now, assume $k \in \{i_2, \ldots, i_{|\mathcal{E}|}\}$ and that (4.1) already has been proven for all preceding events $j \prec k$. We distinguish two cases, depending on which term in Step 3 of algorithm Cpm gets maximal when computing x^{cap}_k.

1. If the first term gets maximal, we have $x_k^{\text{cap}} = \pi_k + d_k \leq x_k^{\text{relax}}$, hence (4.1) holds.

2. If the second term gets maximal, let $\tilde{a}$ denote the activity that in the end causes the delay of event k and let j denote its starting event, i.e.

$$\tilde{a} = (j,k) := \operatorname*{argmax}_{a=(i,k)\in(\mathcal{A}_{\text{train}}\cup\mathcal{A}_{\text{fix}}\cup\mathcal{A}_{\text{head}}^{\text{fix}})} x_i^{\text{cap}} + L_a + d_a.$$

We distinguish three cases, depending on along what type of activity the delay is passed on to event k.

Case 1: If $\tilde{a} \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}}^{\text{fix}}$, then $x_k^{\text{cap}} = x_j^{\text{cap}} + L_{\tilde{a}} + d_{\tilde{a}}$. Using (4.1) to estimate x_j^{cap} and using $x_k^{\text{relax}} - x_j^{\text{relax}} \geq L_{\tilde{a}} + d_{\tilde{a}}$ (as x^{relax} is a feasible timetable, it respects constraints (2.3) for all $a \in \mathcal{A}_{\text{train}}$ and constraints (2.4) for all $a \in \mathcal{A}_{\text{change}}$), we get

$$\begin{aligned}
x_k^{\text{cap}} &= L_{\tilde{a}} + d_{\tilde{a}} + x_j^{\text{cap}} \\
&\leq L_{\tilde{a}} + d_{\tilde{a}} + x_j^{\text{relax}} + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}) \\
&\leq x_k^{\text{relax}} - x_j^{\text{relax}} + x_j^{\text{relax}} + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}) \\
&= x_k^{\text{relax}} + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) + \sum_{\substack{l\in\text{pre}(j):\\ \exists(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m)\in\mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}).
\end{aligned}$$

Using $\text{pre}(j) \subset \text{pre}(k)$ and the fact that each summand of both of the sums is nonnegative, we see that (4.1) is satisfied.

Case 2: If $\tilde{a} \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}$, then $d_{\tilde{a}} = 0$ and $x_k^{\text{cap}} = x_j^{\text{cap}} + L_{jk}$. Using (4.1) to estimate x_j^{cap}, $\pi_k - \pi_j \geq L_{jk}$ (which holds as π is a feasible timetable and $\tilde{a}$ is a forward headway) and $x_k^{\text{relax}} \geq \pi_k$ (which is satisfied due to the fact that x^{relax} is a feasible timetable and hence satisfies (2.2)), we get

$$
\begin{aligned}
x_k^{\text{cap}} &= L_{jk} + x_j^{\text{cap}} \\
&\leq L_{jk} + x_j^{\text{relax}} + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}) \\
&\leq \pi_k - \pi_j + x_j^{\text{relax}} + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}) \\
&\leq x_k^{\text{relax}} - \pi_j + x_j^{\text{relax}} + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}).
\end{aligned}
$$

Again using $\text{pre}(j) \subset \text{pre}(k)$ and the fact that each summand of both of the sums is nonnegative and moving $x_j^{\text{relax}} - \pi_j$ to the first sum proves that (4.1) holds in this case.

Case 3: If $\tilde{a} \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}$, then $d_{\tilde{a}} = 0$ and $x_k^{\text{cap}} = x_j^{\text{cap}} + L_{jk}$. Using (4.1) to estimate x_j^{cap} and adding $x_k^{\text{relax}} \geq 0$ to the right-hand side, we get

$$
\begin{aligned}
x_k^{\text{cap}} &= L_{jk} + x_j^{\text{cap}} \\
&\leq L_{jk} + x_j^{\text{relax}} + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}) \\
&\leq x_k^{\text{relax}} + L_{jk} + x_j^{\text{relax}} + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_l^{\text{relax}} - \pi_l) \\
&\qquad + \sum_{\substack{l \in \text{pre}(j): \\ \exists (l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_l^{\text{relax}} + \max_{(l,m) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{lm}).
\end{aligned}
$$

As before, using $\text{pre}(j) \subset \text{pre}(k)$ and the fact that each summand of both of the sums is nonnegative and moving $x_j^{\text{relax}} + L_{jk}$ to the second sum proves the claim for the third case. □

By using Lemma 4.10, we can prove an upper bound on the relative error of FRFS-FIX if $w_i \geq 1$ for all events $i \in \mathcal{E}$:

Theorem 4.11. *Consider an instance of (DM) with passenger weights $w_i \geq 1$ for all events $i \in \mathcal{E}$. Let $(x^{\text{FRFS-FIX}}, z^{\text{FRFS-FIX}}, g^{\text{FRFS-FIX}})$ be the solution with objective value $F^{\text{FRFS-FIX}}$ computed by heuristic* FRFS-FIX*, and let $(x^{\text{UDM}}, z^{\text{UDM}})$ be the optimal solution of the corresponding instance of (UDM) with objective value F^{UDM}. Furthermore, define*

$$\begin{aligned}
\mathcal{A}_{\text{head}}^{\text{fix}} &:= \left\{(i,j) \in \mathcal{A}_{\text{head}} : g_{ij}^{\text{FRFS-FIX}} = 0\right\} \\
\mathcal{A}_{\text{head}}^{\text{fix-fw}} &:= \mathcal{A}_{\text{head}}^{\text{fix}} \cap \mathcal{A}_{\text{head}}^{\text{forw}} \\
\mathcal{A}_{\text{head}}^{\text{fix-bw}} &:= \mathcal{A}_{\text{head}}^{\text{fix}} \cap \mathcal{A}_{\text{head}}^{\text{back}}.
\end{aligned}$$

Then, the following holds:

a) If $F^{\text{UDM}} \geq 1$, we have

$$\frac{F^{\text{FRFS-FIX}} - F^{\text{UDM}}}{F^{\text{UDM}}} \leq \left(2 + \sum_{\substack{k \in \mathcal{E}: \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (\pi_k + \max_{(k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{kl})\right) \sum_{i \in \mathcal{E}} w_i.$$

b) If x^{UDM} satisfies $\pi_i \leq \pi_j \;\Rightarrow\; x_i^{\text{UDM}} \leq x_j^{\text{UDM}} \quad \forall\, (i,j) \in \mathcal{A}_{\text{head}}$, then

$$\frac{F^{\text{FRFS-FIX}} - F^{\text{UDM}}}{F^{\text{UDM}}} \leq \sum_{i \in \mathcal{E}} w_i.$$

Proof. By construction of the heuristic FRFS-FIX, we have $z^{\text{FRFS-FIX}} = z^{\text{UDM}}$, so we can use the results from Lemma 4.10 with $x^{\text{cap}} := x^{\text{FRFS-FIX}}$ and $x^{\text{relax}} := x^{\text{UDM}}$. Thus, for each event $i \in \mathcal{E}$,

$$
\begin{aligned}
x_i^{\text{FRFS-FIX}} - x_i^{\text{UDM}} \;&\leq \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_k^{\text{UDM}} - \pi_k) + \sum_{\substack{k \in \text{pre}(i) \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_k^{\text{UDM}} + \max_{(k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{kl}) \\
&= \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_k^{\text{UDM}} - \pi_k) + \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (x_k^{\text{UDM}} - \pi_k) \\
&\qquad + \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (\pi_k + \max_{(k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{kl}) \\
&\leq 2F^{\text{UDM}} + \sum_{\substack{k \in \mathcal{E}: \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (\pi_k + \max_{(k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{kl})
\end{aligned}
$$

where the last inequality holds if $w_i \geq 1 \; \forall i \in \mathcal{E}$. As $z^{\text{FRFS-FIX}} = z^{\text{UDM}}$, the second term in the objective of FRFS-FIX and (UDM) is the same, hence

$$
\begin{aligned}
F^{\text{FRFS-FIX}} - F^{\text{UDM}} &= \sum_{i \in \mathcal{E}} w_i (x_i^{\text{FRFS-FIX}} - \pi_i) - \sum_{i \in \mathcal{E}} w_i (x_i^{\text{UDM}} - \pi_i) \\
&= \sum_{i \in \mathcal{E}} w_i (x_i^{\text{FRFS-FIX}} - x_i^{\text{UDM}}) \\
&\leq \left(\sum_{i \in \mathcal{E}} w_i \right) \left(2F^{\text{UDM}} + \sum_{\substack{k \in \mathcal{E}: \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}}} (\pi_k + \max_{(k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-bw}}} L_{kl}) \right).
\end{aligned}
$$

For $F^{\text{UDM}} \geq 1$, claim a) follows directly.

If x^{UDM} satisfies $\pi_i \leq \pi_j \;\Rightarrow\; x_i^{\text{UDM}} \leq x_j^{\text{UDM}} \quad \forall \, (i,j) \in \mathcal{A}_{\text{head}}$, then $\mathcal{A}_{\text{head}}^{\text{fix-bw}} = \emptyset$, so the second sum in (4.1) vanishes, i.e.

$$
x_i^{\text{cap}} \;\leq\; x_i^{\text{relax}} + \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_k^{\text{relax}} - \pi_k)
$$

for all $i \in \mathcal{E}$. Analogously to the proof of claim a), for each event $i \in \mathcal{E}$,

$$
\begin{aligned}
x_i^{\text{FRFS-FIX}} - x_i^{\text{UDM}} &\leq \sum_{\substack{k \in \text{pre}(i): \\ \exists (k,l) \in \mathcal{A}_{\text{head}}^{\text{fix-fw}}}} (x_k^{\text{UDM}} - \pi_k) \\
&\leq F^{\text{UDM}}
\end{aligned}
$$

where the last inequality holds if $w_i \geq 1 \; \forall i \in \mathcal{E}$. Thus,

$$\begin{aligned} F^{\text{FRFS-FIX}} - F^{\text{UDM}} &= \sum_{i \in \mathcal{E}} w_i (x_i^{\text{FRFS-FIX}} - \pi_i) - \sum_{i \in \mathcal{E}} w_i (x_i^{\text{UDM}} - \pi_i) \\ &= \sum_{i \in \mathcal{E}} w_i (x_i^{\text{FRFS-FIX}} - x_i^{\text{UDM}}) \\ &\leq \left(\sum_{i \in \mathcal{E}} w_i \right) F^{\text{UDM}}. \end{aligned}$$

Then, claim b) is a direct consequence. □

The following corollary directly follows from Lemma 4.6:

Corollary 4.12. *The results from Theorem 4.11 also hold for* FRFS.

4.2 Relax & Repair Heuristics

In this section, we suggest different relax & repair heuristics. The advantage of the heuristics which we present is that they can be carried out by hand (at least for small-size instances), without having to solve an NP-hard optimization problem. They all follow a simple concept: First, relax the capacity constraints and apply a given strategy to decide which connections should be maintained, then compute a disposition timetable (for example by applying algorithm CPM). Afterwards, "repair" the solution to get a feasible solution of the capacitated delay management problem (i.e. a solution that respects all capacity constraints). The repair is done as follows: fix all connections and all priority decisions according to the solution of the relaxed problem, i.e. set

$$\mathcal{A}_{\text{fix}} := \{a = (i,j) \in \mathcal{A}_{\text{change}} : x_j - x_i \geq L_a\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\},$$

then solve the corresponding project planning problem (PP($\mathcal{A}_{\text{fix}}$)) to shift the end events of violated headways (and all their successors) into the future.

We suggest three heuristics from this class which apply different strategies when deciding which connections should be maintained and which connections might be dropped and present a worst-case error analysis.

The idea of the first heuristic, NO-WAIT-REPAIR, is to relax the capacity constraints and to apply a *no-wait policy*, i.e. we do not enforce to maintain *any* connection. Then we solve the resulting project planning problem (without connections and without headways), yielding a disposition timetable, and fix the wait/depart decisions and the

priority decisions according to this disposition timetable. In the last step, we solve the resulting project planning problem again (this time with fixed connections and fixed headways) to "repair" the initial solution and to get a disposition timetable that satisfies the fixed headway constraints. Mathematically:

Algorithm 4.5: No-Wait-Repair

Step 1: Set $\mathcal{A}_{\text{fix}} := \emptyset$.

Step 2: Solve (PP($\mathcal{A}_{\text{fix}}$)) and obtain a disposition timetable x.

Step 3: Set $\mathcal{A}_{\text{fix}} := \{a = (i,j) \in \mathcal{A}_{\text{change}} : x_j - x_i \geq L_a\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\}$.

Step 4: Compute the solution of (PP($\mathcal{A}_{\text{fix}}$)).

Note that No-Wait-Repair does not implement a *strict* no-wait policy, but a *non-strict* one: If the lower bound of a changing activity is respected in the disposition timetable of the relaxed problem, the corresponding changing activity is fixed during the repair phase, hence the connection in fact is maintained. However, we do not *enforce* maintaining any connection, but drop all connections that are not maintained "accidentally". In practice, non-strict no-wait policies are used by some public transportation companies.

The strategy for making wait/depart decisions in the next heuristic, All-Wait-Repair, somehow is the opposite of the strategy in No-Wait-Repair: We again relax the capacity constraints, but apply an *all-wait policy*, i.e. we enforce to maintain *all* connections. Then, we solve the resulting project planning problem (with fixed connections and without headways) to get a disposition timetable. In the last step, we use this solution to fix the headway activities according to the disposition timetable and solve the resulting project planning problem again (this time with fixed connections and fixed headways) to "repair" the initial solution. Mathematically:

Algorithm 4.6: All-Wait-Repair

Step 1: Set $\mathcal{A}_{\text{fix}} := \mathcal{A}_{\text{change}}$.

Step 2: Solve (PP($\mathcal{A}_{\text{fix}}$)) and obtain a disposition timetable x.

Step 3: Set $\mathcal{A}_{\text{fix}} := \mathcal{A}_{\text{change}} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\}$.

Step 4: Compute the solution of (PP($\mathcal{A}_{\text{fix}}$)).

The strategy for deciding which connections should be maintained and which connections might be dropped that the third heuristic, Priority-Repair, uses is somewhere in between No-Wait-Repair and All-Wait-Repair: Again, we relax the capacity constraints; but now we maintain only the "most important" connections (similar to Fsfs-Fix). Then we solve the resulting project planning problem (with fixed connections and without headways) to get a disposition timetable which is used to fix the headway activities. In the last step, we again solve the resulting project planning problem (this time with fixed connections and fixed headways) to "repair" violated headway constraints. Mathematically:

Algorithm 4.7: Priority-Repair

Step 1: Maintain the "most important" connections:

a) Sort $\mathcal{A}_{\text{change}}$ in descending order according to the weights w_a.

b) Set $z_a = 0$ for the first $k\%$ of the connections.

Step 2: Set $\mathcal{A}_{\text{fix}} := \{a \in \mathcal{A}_{\text{change}} : z_a = 0\}$.

Step 3: Solve (PP($\mathcal{A}_{\text{fix}}$)) and obtain a disposition timetable x.

Step 4: Set $\mathcal{A}_{\text{fix}} := \{a = (i,j) \in \mathcal{A}_{\text{change}} : x_j - x_i \geq L_a\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : x_i < x_j\}$.

Step 5: Compute the solution of (PP($\mathcal{A}_{\text{fix}}$)).

Note that the first step is the same as in Fsfs-Fix, i.e. both heuristics fix the connections in the same way. However, they differ in how they treat the priority decisions: Fsfs-Fix always fixes the order of two trains as it is in the original timetable, while Priority-Repair fixes the order of two trains like Frfs.

In fact, No-Wait-Repair and All-Wait-Repair are special cases of Priority-Repair (if we set the percentage of "important" connections to either $k = 0$ or $k = 100$); however, as they have practical relevance, we consider them as independent solution approaches.

Note that Frfs-Fix also can be seen as a relax & repair heuristic: In the first step, it also relaxes the headway constraints, but instead of heuristically fixing the wait/depart decisions and solving (PP($\mathcal{A}_{\text{fix}}$)), it directly solves (UDM) to get a solution of the relaxed problem. The last step then is the same as for Priority-Repair: wait/depart decisions and priority decisions are fixed according to the solution of the relaxed problem, then the violated headway constraints are "repaired" by solving (PP($\mathcal{A}_{\text{fix}}$)).

At a first glance, one might think that the solutions computed by FRFS-FIX always are at least as good as the solutions computed by the PRIORITY-REPAIR heuristic (the wait/depart decisions made by PRIORITY-REPAIR always follow a simple rule of thumb; the resulting decisions are only one possible solution that FRFS-FIX takes into account when computing optimal wait/depart decisions for the relaxed problem). However, the solution computed by FRFS-FIX is only at least as good as the one computed by PRIORITY-REPAIR *before* the repair starts; during the recovery, this might switch completely since it might turn out that taking into account the headway constraints, it is better not to maintain a connection that would be maintained in an optimal solution of the uncapacitated problem. We demonstrate this by an example:

Assume a public transportation network with three stations A, B and C. Stations B and C are connected by a single-way track. One train serves line A-B, another train serves line B-C, and a third train serves line C-B. See Figure 4.3 for the corresponding event-activity network.

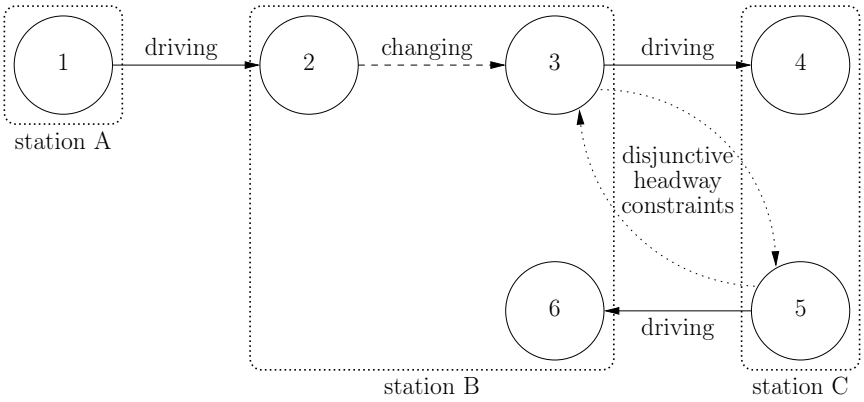

Figure 4.3: In some cases, PRIORITY-REPAIR might yield better solutions than FRFS-FIX.

We assume the lower bounds of all activities to be equal to 2 (i.e. $L_{(1,2)} = L_{(2,3)} = L_{(3,4)} = L_{(5,6)} = 2$), all headways to be equal to 3 ($L_{(3,5)} = L_{(5,3)} = 3$), and all slack times to be equal to 0. Furthermore, we assume that one passenger travels from station A to station C while $T > 1$ passengers travel from station C to station B ($w_4 = w_{(2,3)} = 1$, $w_6 = T$, $w_1 = w_2 = w_3 = w_5 = 0$). Now, let the first train serving the line A-B have a delay of 1 and let $\pi_3 = \pi_2 + 2$, $\pi_5 = \pi_3 + 3$.

In the optimal solution of the uncapacitated problem, the train serving line B-C would wait for the delayed train while the train serving line C-B would be on time, leading to an objective value of 1 – which without doubt is better than the objective value of

T that would be achieved by a no-wait policy. However, if a recovery strategy that does not change the order of trains is applied, the formerly optimal solution gets rather bad: due to the headway, the train serving line C-B also gets a delay of 1, yielding an objective value of $T+1$. Applying the same repair strategy to the solution gained from a no-wait policy does not change anything, so the objective value T of No-Wait-Repair is better than the objective value $T+1$ of Frfs-Fix.

In the following, we show that – similar to the priority-based heuristics presented in Section 4.1 – the relative error of No-Wait-Repair and All-Wait-Repair might get arbitrarily large, depending on the actual input. A result similar to Theorem 4.13 and Theorem 4.14 can be proven for Priority-Repair and for any fixed $k \in]0, 100[$.

The first result deals with all heuristics that solve (DM) or (UDM) by applying a no-wait policy and is not limited to relax & repair heuristics:

Theorem 4.13. *Let* Heu *be a solution approach that solves the delay management problem by applying a no-wait policy. Then, for each $k \in \mathbb{N}$, there exists an instance of (DM) with relative error*

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} > k.$$

Proof. Let $k \in \mathbb{N}$. We show that – even without capacity constraints – the solution computed by a no-wait policy might get arbitrarily bad, only due to "bad" wait/depart decisions. To this end, we use the following example with three stations A, B and C: One train is driving from station A to station B, while another train is driving from station B to station C. Within station B, passengers might transfer from the first train to the second one. All activities have a lower bound of 1 and a slack time of 0. Mathematically, $L_{(1,2)} = L_{(2,3)} = L_{(3,4)} = 1$ and $s_{(1,2)} = s_{(2,3)} = s_{(3,4)} = 0$. Furthermore, let $T = k + 2$ be the common period of all lines. See Figure 4.4 for an illustration.

Now, assume that only one single passenger is traveling in this example, namely from station A to station C, and that the first train has a delay of 1. Furthermore, as we assume slack times of 0, the second train departs one minute after the first train's arrival. Mathematically, $w_1 = w_2 = w_3 = 0$, $w_4 = 1$, $w_{(2,3)} = 1$, $d_{(1,2)} = 1$, and $\pi_3 = \pi_2 + 1$. In an optimal solution, the second train would wait for the delayed train, resulting in a delay of 1 of the passenger at the final station C, yielding $F^{\mathrm{DM}} = 1$. However, if a no-wait policy is applied, the single passenger misses the connection and has to wait for the next train in the next period, yielding $F^{\mathrm{HEU}} = T$ and hence

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} = \frac{T-1}{1} = T - 1 = k + 1 > k. \qquad \square$$

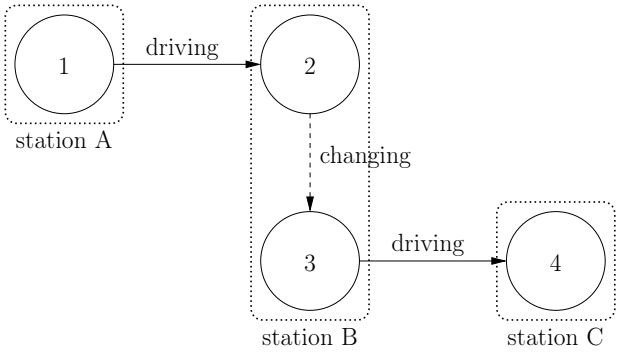

Figure 4.4: Event-activity network for the proof of Theorems 4.13 and 4.14.

The next result deals with all solution approaches that apply an all-wait policy; it is again not limited to relax & repair heuristics, and it again also holds for (UDM):

Theorem 4.14. *Let* Heu *be a solution approach that solves the delay management problem by applying an all-wait policy. Then, for each* $k \in \mathbb{N}$*, there exists an instance of (DM) with relative error*

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} > k.$$

Proof. Let $k \in \mathbb{N}$. We show that – even without capacity constraints – the solution computed by an all-wait policy might get arbitrarily bad, only due to "bad" wait/depart decisions. We use the same example as in the proof of Theorem 4.13, see Figure 4.4 for an illustration. However, we assume different passenger weights: Assume that again one passenger is traveling from station A to station C, but in addition, $(k+1)T$ passengers are traveling from station B to station C. Mathematically, $w_1 = w_2 = w_3 = 0$, $w_4 = 1 + (k+1)T$, $w_{(2,3)} = 1$. In an optimal solution, the second train does not wait for the delayed train, so the passenger traveling from station A to station C misses the connection while all other passengers are on time, yielding an objective value of $F^{\mathrm{DM}} = T$. However, if an all-wait strategy is applied, all passengers have a delay of 1 when arriving at station C, yielding $F^{\mathrm{HEU}} = 1 + (k+1)T$, hence

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} = \frac{1 + (k+1)T - T}{T} = \frac{1 + kT}{T} > k. \qquad \square$$

In the proofs of Theorem 4.13 and Theorem 4.14, the arbitrarily high relative error has been caused only by "bad" wait/depart decisions. However, ignoring headway constraints, solving the relaxed (uncapacitated) problem and applying a repair strategy afterwards also might lead to an arbitrarily high relative error, independently of wait/depart decisions:

Theorem 4.15. *Let* HEU *be a solution approach that solves the capacitated delay management problem by relaxing the capacity constraints, solving the relaxed problem, and applying a repair strategy. Then, for each $k \in \mathbb{N}$, there exists an instance of (DM) with relative error*

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} > k.$$

Proof. Let $k \in \mathbb{N}$. We consider the simple case of two stations A and B, connected by a single-way track. See Figure 4.5 for an illustration of the corresponding event-activity network.

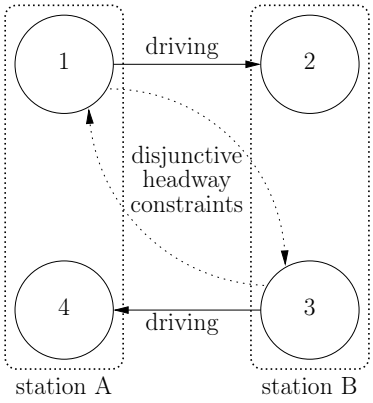

Figure 4.5: Event-activity network for the proof of Theorem 4.15.

Now, assume that the train driving from station A to station B is a very fast train with a driving time of 1 (hence the corresponding headway $L_{(1,3)} = 2$ is very small), while the train driving from station B to station A is a very slow train with a driving time of $6k + 9$ (hence the corresponding headway $L_{(3,1)} = 6k + 10$ is very large). Assume that in the original timetable, the fast train drives first, e.g., $\pi_3 = \pi_1 + 2$. Both trains carry one passenger ($w_2 = w_4 = 1$). Furthermore, assume that the departure of the fast train is delayed by $d_1 = 3$.

In an optimal solution, the fast train drives first, despite of its delay. The slow train has to wait for the fast train, so both trains arrive with a delay of 3 at their final destination, yielding an objective value of $F^{\mathrm{DM}} = 6$.

However, if capacity constraints are neglected, the slow, punctual train drives first and arrives at station A on time. The fast train has to wait while the track is blocked by the slow train – during the recovery phase, the departure of the fast train therefore is

re-scheduled in such a way that it finally has a delay of $6k + 10 + 2$. This leads to an objective value of $F^{\mathrm{HEU}} = 6k + 12$, hence

$$\frac{F^{\mathrm{HEU}} - F^{\mathrm{DM}}}{F^{\mathrm{DM}}} = \frac{6k + 12 - 6}{6} = k + 1 > k. \qquad \square$$

To prove that the relative error might get arbitrarily large, we constructed artificial examples with arbitrarily large period lengths, passenger weights, or headway times. Other possible approaches for proving Theorems 4.8, 4.9, 4.13, 4.14, and 4.15 are based on making the network larger such that delays propagate through the whole network and affect more and more events. However, all these approaches lead to artificial problem instances that are not very likely to appear in practice – but they show that general worst-case error bounds for the heuristics presented in this paper either have to depend on the actual input instance (at least on the period length, passenger weights, headway times, network size, etc.), or only can be proven for a limited set of input instances.

Despite of the worst-case results, the solution approaches which we have suggested in Sections 4.1 and 4.2 behave quite well *in average* as we show in the next section.

4.3 Numerical Results

To confirm the effectiveness of the different solution approaches suggested in this chapter and to show that the results on the relative error are only worst-case results, we again present results from our case study based on real-world data. The underlying data set is the same as described in Section 3.5.

4.3.1 Influence of the Percentage of Fixed Connections on the Relative Error of Fsfs-Fix and Priority-Repair

First, we analyze the influence of the parameter k (i.e. the percentage of fixed "important" connections). To this end, we compare the relative error of different variants of Fsfs-Fix and Priority-Repair, varying only in the parameter k. For our analysis, we use five different values $k \in \{0, 25, 50, 75, 100\}$ where $k = 0$ represents a no-wait policy and $k = 100$ an all-wait policy.

As it turned out during the case study, values of k which yield good results for small source delays lead to large relative errors for large source delays and vice versa. Hence we combine different variants of Fsfs-Fix in an approach called Best-Fsfs-Fix: For each input instance, run Fsfs-Fix with different values for k and for each single scenario,

take the best solution. Analogously, Best-Repair denotes a similar approach based on Priority-Repair. Formally:

Algorithm 4.8: Best-Fsfs-Fix

Input: An instance i of (DM).

Step 1: Solve i by running Fsfs-Fix with $k \in \{0, 25, 50, 75, 100\}$ and obtain solutions (x^k, z^k, g) with objective values F^k.

Step 2: Let $k^* = \underset{k \in \{0,25,50,75,100\}}{\operatorname{argmin}} F^k$.

Output: A solution (x^{k^*}, z^{k^*}, g) for i with objective value F^{k^*}.

Algorithm 4.9: Best-Repair

Input: An instance i of (DM).

Step 1: Solve i by running Priority-Repair with $k \in \{0, 25, 50, 75, 100\}$ and obtain solutions (x^k, z^k, g^k) with objective values F^k.

Step 2: Let $k^* = \underset{k \in \{0,25,50,75,100\}}{\operatorname{argmin}} F^k$.

Output: A solution $(x^{k^*}, z^{k^*}, g^{k^*})$ for i with objective value F^{k^*}.

We analyze three classes of delay scenarios: For the class "small source delays", in each scenario, we randomly generated ten source delays between 60 and 180 seconds. The class "large source delays" contains scenarios in which we generated ten source delays between 1 500 and 1 800 seconds. All scenarios in the class "mixed source delays" feature ten randomly generated source delays between 180 and 900 seconds (as in Section 3.5).

For each class of delay scenarios and for each observation period, we generated about 400 different delay scenarios and solved each resulting instance of the delay management problem both exactly and by invoking each of the heuristics presented in this chapter.

We start with analyzing Fsfs-Fix and Priority-Repair on the class of small source delays. The relative errors for different variants of Fsfs-Fix are summarized in Table 4.6, the average relative error is depicted in Figure 4.7. For all observation periods, both the maximal and the average relative error decrease when k grows. For Priority-Repair, the relative errors are given in Table 4.8 while the average relative error also

is illustrated in Figure 4.9. For all observation periods, the average relative error of Priority-Repair decreases with growing k, too, and apart from some exceptions, the same is true for its maximal relative error. Hence, for the data set of our case study, the best fixed rule when small source delays occur is an all-wait policy.

If we consider large source delays, the situation changes radically. For this case, the relative errors of Fsfs-Fix and Priority-Repair are summarized in Tables 4.10 and 4.12, while the average relative errors are depicted in Figures 4.11 and 4.13. For both heuristics and for each observation period, the average relative errors grow with k, and with only two exceptions, the same holds for the maximal relative errors. Hence, for the data set of our case study, the best fixed rule when large source delays occur is a no-wait policy.

Finally, we compare different variants of Fsfs-Fix and Priority-Repair on the third class of delay scenarios. We present an overview of the relative errors in Table 4.14 and Table 4.16 and depict the average relative errors in Figure 4.15 and Figure 4.17. As in the case of large source delays, both the average and the maximal relative errors increase when k grows. Hence, again a no-wait strategy yields the lowest average relative error and minimizes the maximal relative error, too.

Due to the fact that the parameter k has such a large effect on the relative error (and since this effect in the case of small source delays is in diametrical opposition to the effect in the case of large source delays), it is reasonable not to use one fixed variant of Fsfs-Fix or Priority-Repair, but to use the Best-Fsfs-Fix or Best-Repair approach. Hence, when comparing Fsfs-Fix and Priority-Repair with other heuristics in Section 4.3.2 and in Section 4.3.3, we always use Best-Fsfs-Fix and Best-Repair instead of Fsfs-Fix and Priority-Repair.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	3.7100	0.2169	0.4337	0.0319	0.4151	0.0204
4	2.1139	0.1870	0.5945	0.0301	0.3085	0.0170
6	1.8266	0.1964	0.6300	0.0415	0.3791	0.0255
8	2.8212	0.1731	0.4453	0.0321	0.3340	0.0185
10	1.6615	0.1953	0.6218	0.0459	0.3257	0.0193
12	2.0049	0.1854	0.7937	0.0341	0.7937	0.0220

	$k = 75$		$k = 100$		Best-Fsfs-Fix	
hours	max	avg	max	avg	max	avg
2	0.4151	0.0103	0.4151	0.0046	0.4151	0.0042
4	0.2293	0.0050	0.2293	0.0040	0.2293	0.0035
6	0.3791	0.0098	0.3791	0.0079	0.3791	0.0077
8	0.2597	0.0075	0.2060	0.0042	0.2060	0.0039
10	0.3257	0.0087	0.3257	0.0063	0.3257	0.0053
12	0.5814	0.0118	0.5814	0.0094	0.5814	0.0083

Table 4.6: Average and maximal relative error of Fsfs-Fix for small source delays.

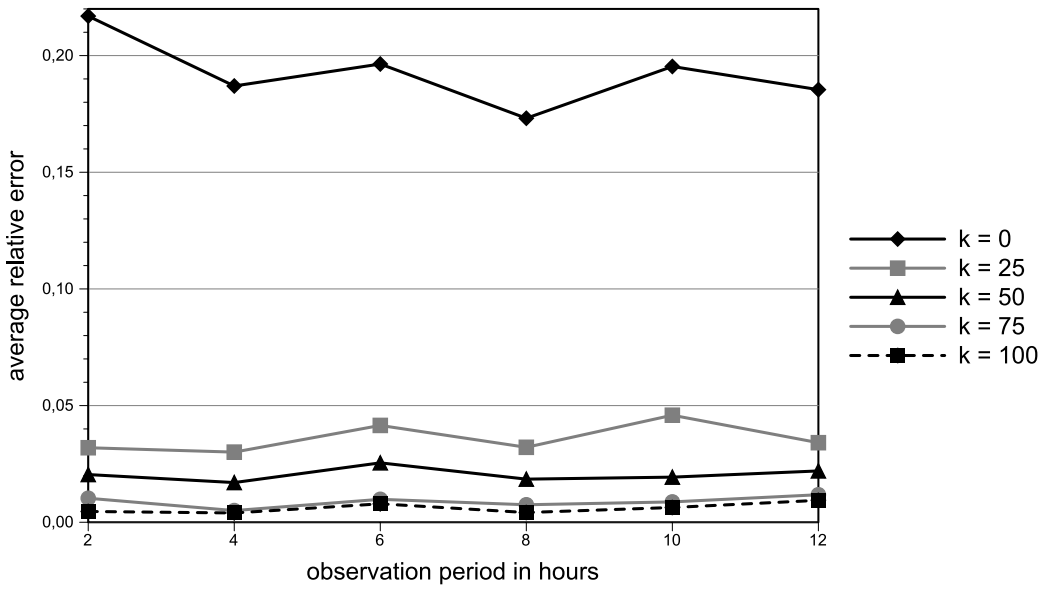

Figure 4.7: Average relative error of Fsfs-Fix, depending on the size of the observation period, for small source delays.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	4.7921	0.3357	4.7921	0.1309	2.8715	0.0830
4	6.1489	0.3829	6.1383	0.2055	6.1383	0.1794
6	6.6541	0.3679	6.0740	0.2020	6.0740	0.1822
8	9.0051	0.5140	8.7283	0.3533	6.5679	0.1841
10	12.9513	0.3043	12.9513	0.1485	12.9513	0.1211
12	8.8329	0.3238	7.3069	0.1562	7.3069	0.1360

	$k = 75$		$k = 100$		Best-Repair	
hours	max	avg	max	avg	max	avg
2	2.8715	0.0729	2.8715	0.0672	2.8715	0.0663
4	6.1383	0.1674	6.1383	0.1663	6.1383	0.1655
6	8.5783	0.1731	8.9225	0.1721	6.0740	0.1637
8	6.5679	0.1731	6.5679	0.1698	6.5679	0.1695
10	12.9513	0.1106	12.9513	0.1082	12.9513	0.1058
12	10.3362	0.1324	10.7562	0.1314	7.3069	0.1223

Table 4.8: Average and maximal relative error of Priority-Repair for small source delays.

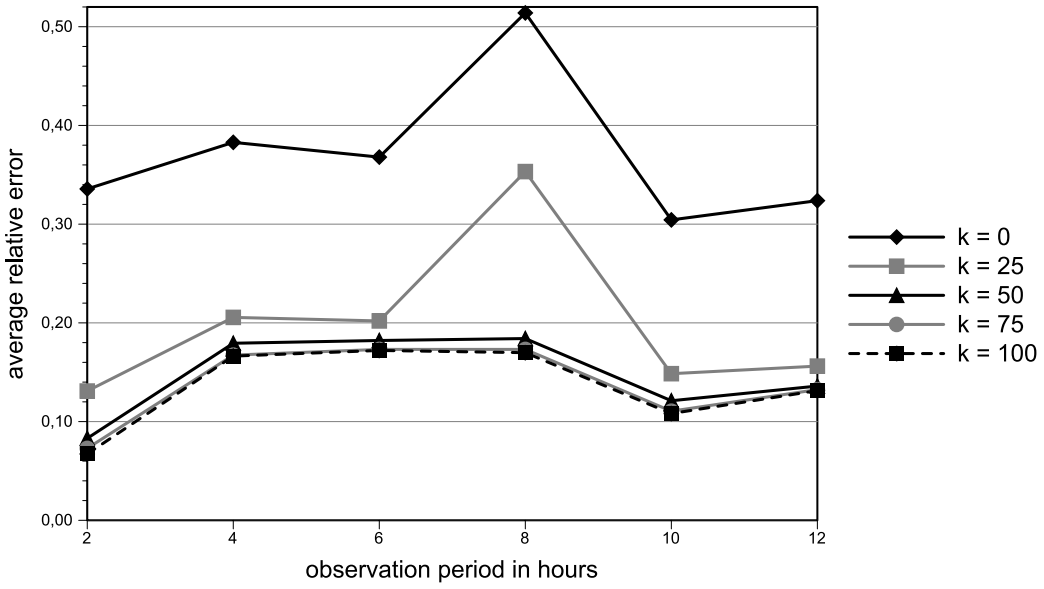

Figure 4.9: Average relative error of Priority-Repair, depending on the size of the observation period, for small source delays.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	2.5852	0.3921	2.5688	0.4270	4.2746	0.4748
4	4.8944	0.9151	5.2519	1.0235	5.7679	1.1799
6	6.6746	1.2786	7.1859	1.4408	8.3953	1.7256
8	8.9313	1.4631	9.7857	1.7034	10.2950	2.0201
10	12.9596	1.4490	13.3854	1.6730	14.4160	2.0317
12	11.2851	1.4867	11.9408	1.7509	14.5262	2.0686

	$k = 75$		$k = 100$		Best-Fsfs-Fix	
hours	max	avg	max	avg	max	avg
2	4.2746	0.4996	4.5889	0.5272	2.5677	0.3843
4	5.9564	1.2830	6.0409	1.3292	4.8944	0.9104
6	8.6063	1.9001	8.7733	1.9886	6.6746	1.2728
8	13.5434	2.2881	14.0488	2.3845	8.9313	1.4579
10	16.5247	2.3475	17.4129	2.4824	12.9596	1.4444
12	20.2093	2.4272	20.3932	2.5746	11.2851	1.4812

Table 4.10: Average and maximal relative error of Fsfs-Fix for large source delays.

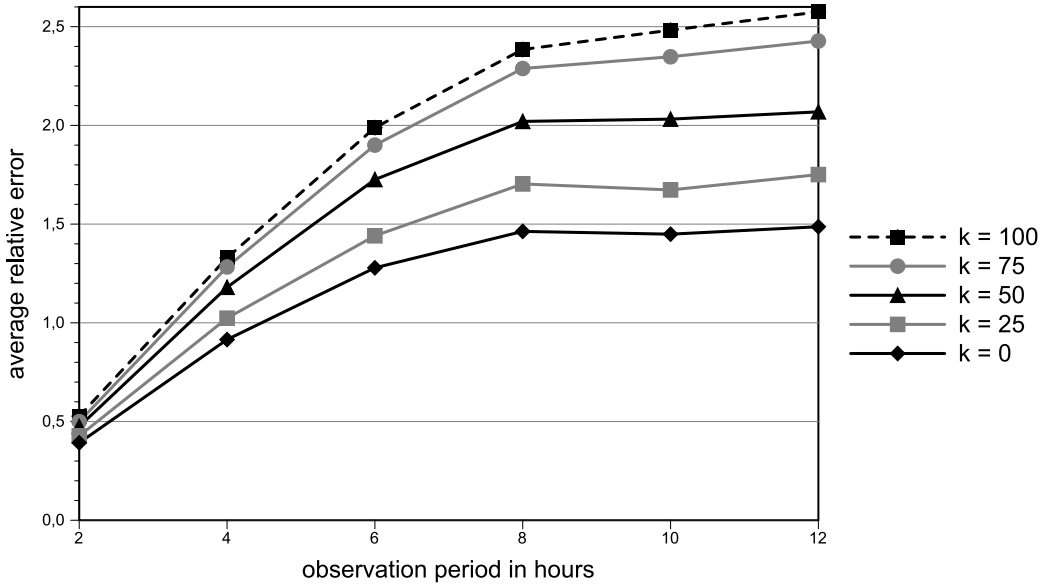

Figure 4.11: Average relative error of Fsfs-Fix, depending on the size of the observation period, for large source delays.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	1.2704	0.0675	1.2704	0.1095	1.8971	0.1654
4	6.9360	0.2546	7.6836	0.3988	8.6981	0.5761
6	13.3661	0.8108	14.4953	1.0696	22.4031	1.4974
8	30.0758	1.5590	33.6386	1.9273	39.2582	2.6344
10	40.8221	1.9558	42.3855	2.3405	45.6342	3.1980
12	36.0509	2.0466	39.1093	2.4290	42.9223	3.3764

	$k = 75$		$k = 100$		Best-Repair	
hours	max	avg	max	avg	max	avg
2	2.3358	0.2300	2.7501	0.2718	1.2499	0.0638
4	10.0314	0.7724	10.0764	0.8908	6.9360	0.2518
6	24.9630	1.8892	26.6913	2.1468	13.3661	0.8106
8	39.5246	3.2449	40.5419	3.6115	30.0758	1.5327
10	42.2398	4.0302	46.5789	4.4183	33.6390	1.8997
12	88.5609	4.5261	89.7079	5.0700	36.0509	1.9356

Table 4.12: Average and maximal relative error of Priority-Repair for large source delays.

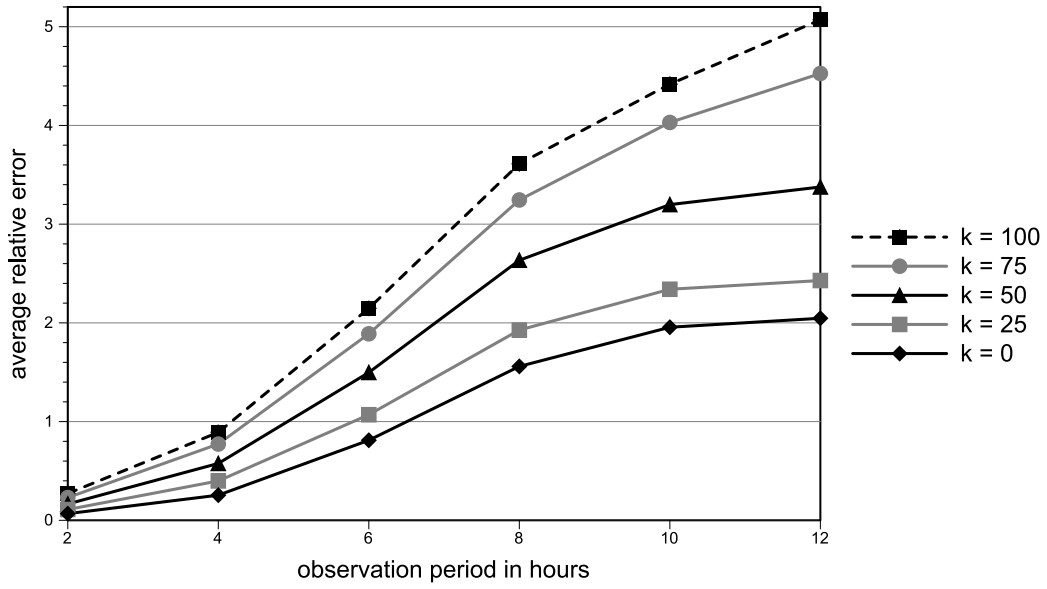

Figure 4.13: Average relative error of Priority-Repair, depending on the size of the observation period, for large source delays.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	1.6042	0.1758	1.6336	0.1494	1.8881	0.1680
4	1.8154	0.2265	1.7595	0.2282	1.7510	0.2500
6	1.9983	0.2391	1.9786	0.2518	2.3513	0.2864
8	2.2311	0.2369	2.6057	0.2411	2.6869	0.2838
10	1.3684	0.2542	1.6049	0.2676	1.8055	0.2929
12	4.0549	0.2584	4.7230	0.2771	5.1889	0.2960

	$k = 75$		$k = 100$		Best-Fsfs-Fix	
hours	max	avg	max	avg	max	avg
2	1.8881	0.1807	1.9070	0.1840	1.6042	0.1410
4	1.9851	0.2684	1.9851	0.2738	1.7510	0.2050
6	2.3513	0.3146	2.3513	0.3169	1.9786	0.2184
8	2.7407	0.3058	2.7642	0.3099	2.2311	0.2131
10	2.2144	0.3199	2.2314	0.3238	1.2877	0.2298
12	5.2352	0.3219	5.2352	0.3254	4.0549	0.2373

Table 4.14: Average and maximal relative error of Fsfs-Fix for mixed source delays.

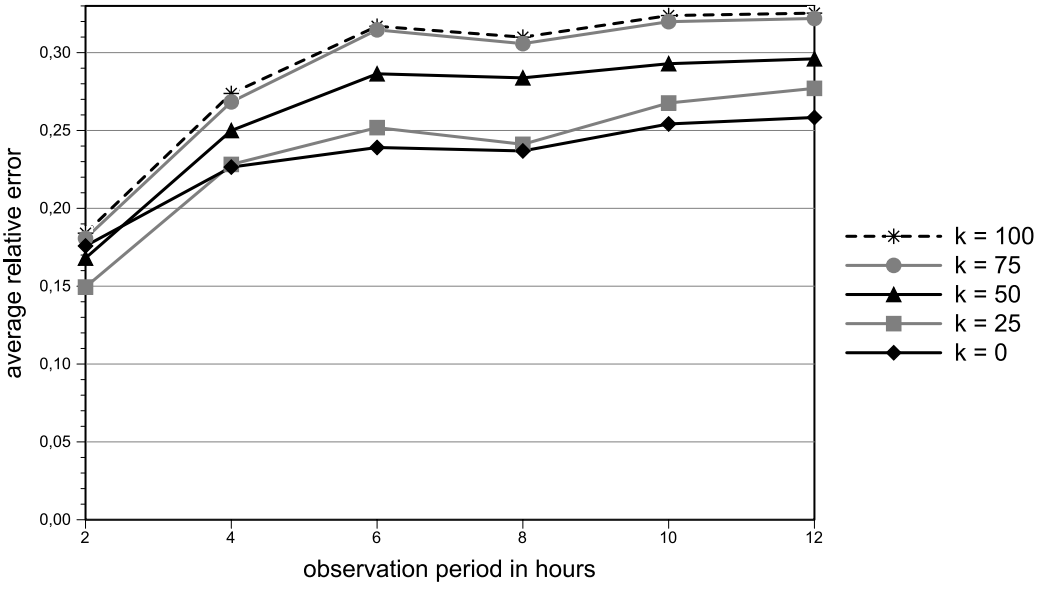

Figure 4.15: Average relative error of Fsfs-Fix, depending on the size of the observation period, for mixed source delays.

	$k = 0$		$k = 25$		$k = 50$	
hours	max	avg	max	avg	max	avg
2	1.0800	0.1237	1.0187	0.0987	1.5671	0.0913
4	7.7600	0.3161	7.7646	0.3297	7.7622	0.3429
6	47.7743	0.8774	49.4055	0.9417	52.8792	1.0278
8	150.5909	2.0991	155.7212	2.2107	168.3575	2.3881
10	113.1986	1.0459	116.7790	1.1275	125.8791	1.2727
12	271.3629	1.3865	280.3381	1.4734	299.4943	1.5723

	$k = 75$		$k = 100$		Best-Repair	
hours	max	avg	max	avg	max	avg
2	1.7714	0.1180	1.7714	0.1324	0.7567	0.0548
4	7.7622	0.4198	7.8109	0.4350	7.7600	0.2418
6	53.8926	1.2053	54.0041	1.2221	47.7743	0.7844
8	173.1206	2.6307	174.1113	2.6767	150.5909	2.0010
10	131.4135	1.5572	131.9608	1.5972	113.1986	0.9568
12	315.5096	1.9773	316.6856	2.0038	271.3629	1.2905

Table 4.16: Average and maximal relative error of Priority-Repair for mixed source delays.

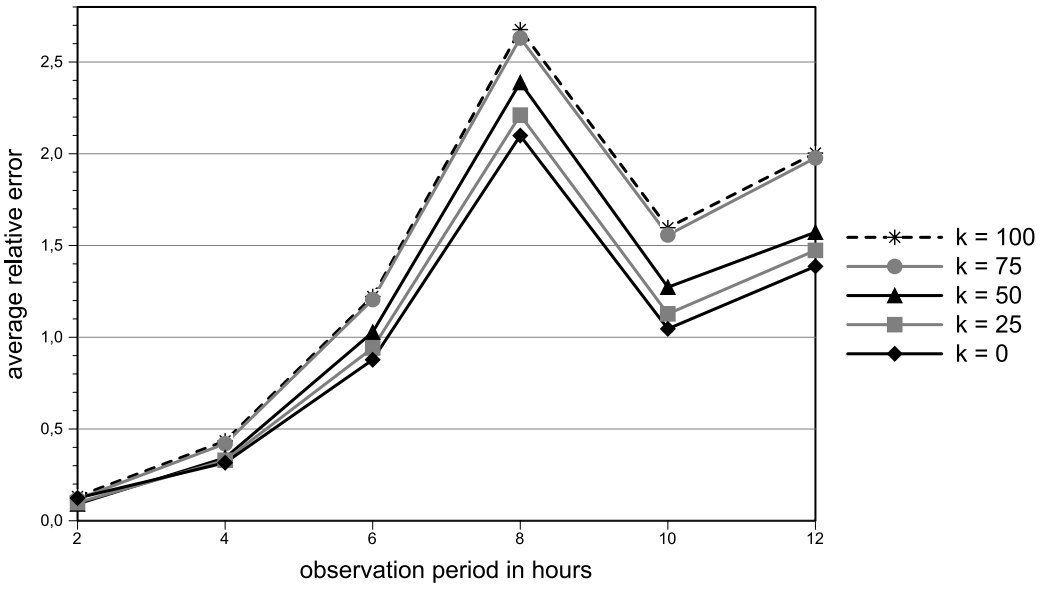

Figure 4.17: Average relative error of Priority-Repair, depending on the size of the observation period, for mixed source delays.

4.3.2 Computation Times

Now we compare the computation times of different solution approaches. As representatives for FSFS-FIX and PRIORITY-REPAIR, we consider the BEST-FSFS-FIX and BEST-REPAIR approaches introduced in Section 4.3.1. Similar to those approaches, BEST-POLY denotes the approach of running BEST-FSFS-FIX and BEST-REPAIR and taking the best solution for each instance, while for BEST-ALL, we run *all* solution procedures suggested in this chapter (except for FRFS-FIX and FSFS-FIX which are, according to Lemma 4.6 and Lemma 4.7, never better than FRFS and FSFS) and take the best solution. Formally:

Algorithm 4.10: BEST-POLY

Input: An instance i of (DM).

Step 1: Solve i by running BEST-FSFS-FIX and obtain a solution (x^1, z^1, g^1) with objective value F^1.

Step 2: Solve i by running BEST-REPAIR and obtain a solution (x^2, z^2, g^2) with objective value F^2.

Step 3: Let $k^* = \operatorname{argmin}_{k \in \{1,2\}} F^k$.

Output: A solution $(x^{k^*}, z^{k^*}, g^{k^*})$ for i with objective value F^{k^*}.

Algorithm 4.11: BEST-ALL

Input: An instance i of (DM).

Step 1: Solve i by running FSFS and obtain a solution (x^1, z^1, g^1) with objective value F^1.

Step 2: Solve i by running FRFS and obtain a solution (x^2, z^2, g^2) with objective value F^2.

Step 3: Solve i by running BEST-REPAIR and obtain a solution (x^3, z^3, g^3) with objective value F^3.

Step 4: Let $k^* = \operatorname{argmin}_{k \in \{1,2,3\}} F^k$.

Output: A solution $(x^{k^*}, z^{k^*}, g^{k^*})$ for i with objective value F^{k^*}.

As it turned out that the computation times show a similar behavior for each of the three classes of delay scenarios, in this section, we focus on the class of mixed source delays. The computation times for all heuristics, compared to the time needed for computing an optimal solution, are given in Table 4.18. As can be seen there, even in the worst case, the computation time of Fsfs, Frfs, and Frfs-Fix never exceeds 30% of the computation time needed for solving the initial problem to optimality, and for larger observation periods (when computing an optimal solution takes rather long and heuristic solution approaches get more important), the relative computation time is even reduced to about 0.3%-0.5% (which means a speed-up of factor 200-300). The results for Best-Fsfs-Fix and Best-Repair are similar, and even Best-All and Best-Poly (for which we have to solve each instance 7 and 10 times, each time using a different algorithm) reduce the computation time for larger observation periods to an average of 1-2%. As Best-All has to solve several hard problems, while Best-Poly only makes use of heuristics with guaranteed polynomial runtime, for larger observation periods, Best-Poly is faster than Best-All.

	Fsfs		Frfs		Frfs-Fix	
hours	max	avg	max	avg	max	avg
2	0.1667	0.0386	0.2955	0.0731	0.2955	0.0685
4	0.0461	0.0123	0.0850	0.0222	0.0752	0.0187
6	0.1189	0.0043	0.1962	0.0072	0.1227	0.0053
8	0.1785	0.0031	0.2550	0.0051	0.1445	0.0034
10	0.1099	0.0030	0.2064	0.0047	0.1132	0.0031
12	0.0780	0.0031	0.1310	0.0048	0.1004	0.0029

	Best-Fsfs-Fix		Best-Repair		Best-Poly		Best-All	
hours	max	avg	max	avg	max	avg	max	avg
2	0.7045	0.1668	0.6818	0.1512	1.3864	0.3180	1.1364	0.2630
4	0.1917	0.0455	0.1578	0.0391	0.3495	0.0846	0.2888	0.0736
6	0.2801	0.0130	0.2338	0.0100	0.5139	0.0230	0.3483	0.0145
8	0.3088	0.0082	0.2351	0.0064	0.5439	0.0147	0.6686	0.0147
10	0.2948	0.0075	0.1993	0.0053	0.4941	0.0128	0.4976	0.0131
12	0.2446	0.0068	0.1846	0.0048	0.4292	0.0116	0.3770	0.0128

Table 4.18: Average and maximal relative computation time of different heuristic solution approaches.

4.3.3 Relative Errors

We conclude the presentation of the results of the case study by comparing the relative errors of all solution approaches suggested in this chapter. To this end, we again consider the three different classes of delay scenarios from Section 4.3.1. In Table 4.19, we show the relative errors of all heuristics for the case of small source delays; the average relative errors are depicted in Figure 4.20. Obviously, if only small source delays occur, all heuristics which fix the order of trains according to the original timetable (Fsfs, Best-Fsfs-Fix) or which make use of such a heuristic (Best-Poly, Best-All) have quite small average relative errors; in addition, their maximal relative errors are rather small, too. By contrast, all heuristics which might change the order of trains (Frfs, Frfs-Fix, Best-Repair) have significantly larger average relative errors for all observation periods, and their maximal relative errors get quite large. It is remarkable that the average relative errors of Fsfs, Best-Fsfs-Fix, Best-Poly, and Best-All are very close together for all observation periods as can be seen in Figure 4.20. The same holds for Frfs, Frfs-Fix, Best-Repair. Probably priority decisions have a much greater influence on the relative error than wait/depart decisions have as long as only small source delays are considered.

For the case of large source delays, the relative errors are shown in Table 4.21, while the average relative errors are also presented in Figure 4.22. For short observation periods (up to six hours), in contrast to the case of small source delays, all heuristics which change the order of trains (Frfs, Frfs-Fix, Best-Repair) or which make use of such a heuristic (Best-Poly, Best-All) have smaller relative errors than the heuristics which fix the order of trains according to the original timetable (Fsfs, Best-Fsfs-Fix). However, taking into account long-term effects (i.e. larger observation periods), Fsfs and Best-Fsfs-Fix again have smaller average and maximal relative errors than Frfs, Frfs-Fix, and Best-Repair. Note that there is a large gap between the relative error of Frfs and Frfs-Fix (both for the average and for the maximal relative error), while Frfs and Best-Repair are close together (again concerning both, the maximal and the average relative error). Thus, if the order of trains is swapped, wait/depart decisions have a significant influence on the relative error. The average relative errors of Fsfs and Best-Fsfs-Fix as well as the maximal relative errors of both approaches are again close together; hence in the scenarios considered here, making optimal wait/depart decisions does not lead to significantly better results than choosing the best solution from several solutions which are based on fixing different subsets of all connections.

Finally, in Table 4.23, we summarize the relative errors for the case of mixed delays and depict the average relative error in Figure 4.24. Note that for all solution approaches which either fix the order of trains as it is in the original timetable (Fsfs, Best-Fsfs-Fix) or which make use of these heuristics (Best-Poly, Best-All), the average and maximal relative errors are rather small. By contrast, all heuristics which fix the order of

trains according to the solution of the corresponding uncapacitated delay management problem (Frfs, Frfs-Fix) as well as Best-Repair yield very large relative errors and rather large average relative errors as already shown in Section 4.3.1 (but fortunately only in few cases). As in the case of large source delays, the maximal and the average relative errors of Fsfs and Best-Fsfs-Fix are very close to each other. The same holds for Frfs and Best-Repair, while there is again a large gap between Frfs and Frfs-Fix.

In practice, it is not only important how large the relative error is in average or in the worst case, but also how often a solution procedure yields a solution that is "close enough" to the optimal one. Thus we show how often the objective value of each heuristic is "close" to the optimal solution in Tables 4.25-4.30. In the case of small source delays, the number of affected events is rather limited, due to the slack times of the driving activities, and a "bad" wait/depart decision has only little influence on the objective. This is also reflected by the results in Table 4.25 and Table 4.26 where we summarize in how many cases the objective value of a heuristic solution is at most 105% of (or equal to) the objective value of an optimal solution. In Table 4.19 and Figure 4.20, we already showed that in the case of small source delays, Fsfs and Best-Fsfs-Fix have smaller maximal and average relative errors than Frfs, Frfs-Fix, and Best-Repair. In accordance with this fact, the number of cases in which Fsfs or Best-Fsfs-Fix are "close" to the optimum is larger than for Frfs, Frfs-Fix, and Best-Repair as can be seen in Table 4.25 and Table 4.26. Hence, in the case of small source delays, in most cases the best strategy is not to change the order of trains.

In the case of large source delays, things totally change. In Table 4.27 and Table 4.28, we summarize in how many cases the objective value of the different heuristic approaches is at most 120% (105%) of the objective value of an optimal solution. The number of cases in which Frfs, Frfs-Fix, or Best-Repair are "close" to the optimum is larger than for Fsfs or Best-Fsfs-Fix. This is consistent with the results presented in Table 4.21 and Figure 4.22 where it turned out that for large source delays, the maximal and average relative errors of Frfs, Frfs-Fix, and Best-Repair are lower than those of Fsfs and Best-Fsfs-Fix. These results suggest that in the case of large source delays, changing the order of trains is a good strategy in many cases – however, in some cases, this strategy leads to very large relative errors as can be seen in Table 4.21.

In Table 4.29 and Table 4.30, we summarize in how many cases the objective value of the different heuristic approaches is at most 105% (101%) of the objective value of an optimal solution if we consider mixed delay scenarios. As for large source delays, the number of cases in which Frfs, Frfs-Fix, or Best-Repair are "close" to the optimum is larger than for Fsfs or Best-Fsfs-Fix, although their maximal and average relative errors are significantly higher especially for larger observation periods, see Table 4.23. However, the difference is not as large as in the case of large source delays.

	Fsfs		Frfs		Frfs-Fix	
hours	max	avg	max	avg	max	avg
2	0.4151	0.0031	2.8715	0.0669	2.8715	0.0680
4	0.2293	0.0035	6.1531	0.1711	6.1531	0.1711
6	0.3791	0.0066	6.0740	0.1691	8.9225	0.1784
8	0.2060	0.0031	6.5679	0.1741	6.5679	0.1750
10	0.3257	0.0052	12.9513	0.1076	12.9513	0.1091
12	0.5814	0.0075	7.3069	0.1232	10.7562	0.1322

	Best-Fsfs-Fix		Best-Repair		Best-Poly		Best-All	
hours	max	avg	max	avg	max	avg	max	avg
2	0.4151	0.0042	2.8715	0.0663	0.4151	0.0029	0.4151	0.0023
4	0.2293	0.0035	6.1383	0.1655	0.2293	0.0026	0.2293	0.0026
6	0.3791	0.0077	6.0740	0.1637	0.3791	0.0066	0.3791	0.0055
8	0.2060	0.0039	6.5679	0.1695	0.2060	0.0038	0.2060	0.0030
10	0.3257	0.0053	12.9513	0.1058	0.3257	0.0039	0.3257	0.0038
12	0.5814	0.0083	7.3069	0.1223	0.3437	0.0060	0.3437	0.0053

Table 4.19: Average and maximal relative error of different heuristic solution approaches for small source delays.

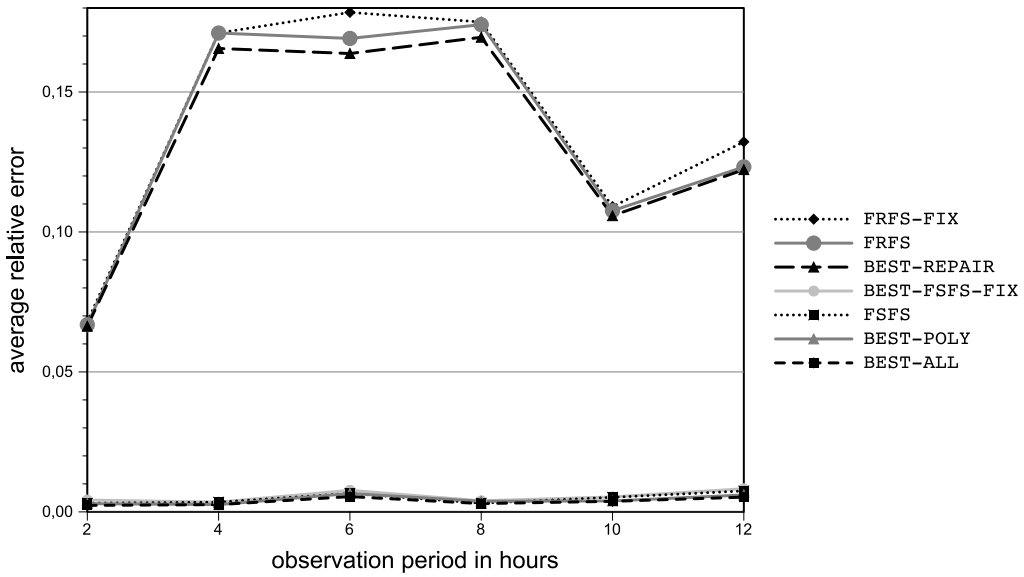

Figure 4.20: Average relative error of different heuristic solution approaches, depending on the size of the observation period, for small source delays.

	Fsfs		Frfs		Frfs-Fix	
hours	max	avg	max	avg	max	avg
2	2.5677	0.3776	1.2659	0.0628	1.3363	0.0672
4	4.8346	0.8946	6.9360	0.2611	9.4894	0.3796
6	6.6261	1.2497	13.2704	0.8152	22.8606	1.2672
8	8.8660	1.4322	30.0354	1.5537	75.7176	2.5260
10	12.8747	1.4191	40.6974	1.9599	46.9756	2.9999
12	11.2213	1.4561	35.9256	2.0455	89.3005	3.3618

	Best-Fsfs-Fix		Best-Repair		Best-Poly		Best-All	
hours	max	avg	max	avg	max	avg	max	avg
2	2.5677	0.3843	1.2499	0.0638	1.0704	0.0566	1.0704	0.0543
4	4.8944	0.9104	6.9360	0.2518	2.5720	0.1916	2.5720	0.1861
6	6.6746	1.2728	13.3661	0.8106	4.0459	0.3521	4.0119	0.3426
8	8.9313	1.4579	30.0758	1.5327	5.5770	0.4263	5.5017	0.4170
10	12.9596	1.4444	33.6390	1.8997	7.8935	0.4376	7.8068	0.4263
12	11.2851	1.4812	36.0509	1.9356	8.7683	0.3920	8.6541	0.3817

Table 4.21: Average and maximal relative error of different heuristic solution approaches for large source delays.

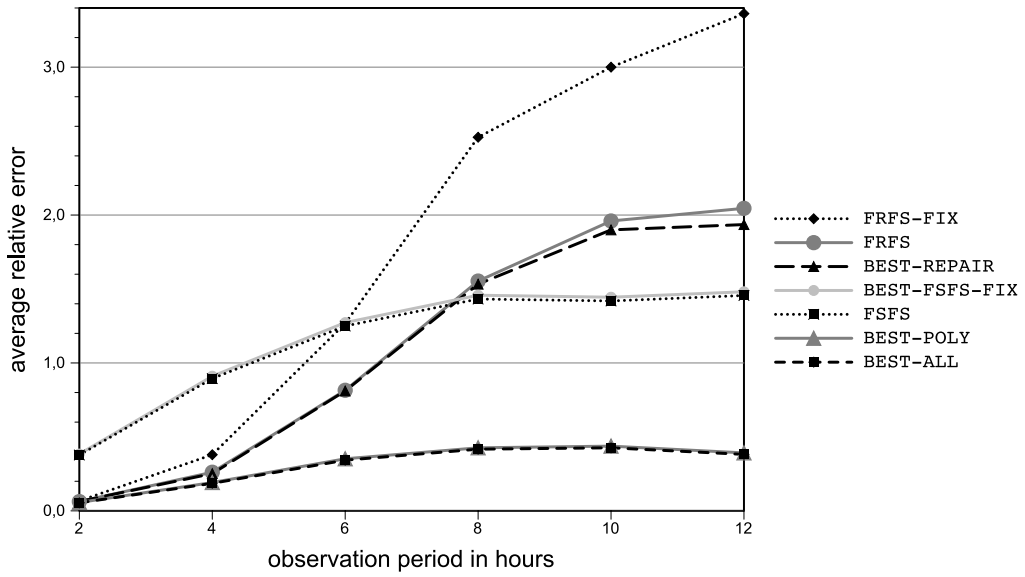

Figure 4.22: Average relative error of different heuristic solution approaches, depending on the size of the observation period, for large source delays.

	Fsfs		Frfs		Frfs-Fix	
hours	max	avg	max	avg	max	avg
2	1.6029	0.1379	0.7638	0.0743	0.7638	0.0760
4	1.7039	0.1976	8.3164	0.2558	8.5055	0.3099
6	1.9680	0.2092	47.7114	0.7983	54.0041	0.9871
8	2.1844	0.2037	150.0306	2.0189	174.0071	2.4342
10	1.2727	0.2208	112.8261	0.9775	131.8755	1.2637
12	3.9937	0.2273	270.5514	1.3131	316.5384	1.6090

	Best-Fsfs-Fix		Best-Repair		Best-Poly		Best-All	
hours	max	avg	max	avg	max	avg	max	avg
2	1.6042	0.1410	0.7567	0.0548	0.3984	0.0350	0.3984	0.0317
4	1.7510	0.2050	7.7600	0.2418	0.8798	0.0931	0.8795	0.0844
6	1.9786	0.2184	47.7743	0.7844	1.6711	0.1068	1.6515	0.0978
8	2.2311	0.2131	150.5909	2.0010	0.8815	0.1057	0.8736	0.0966
10	1.2877	0.2298	113.1986	0.9568	0.8043	0.1057	0.7822	0.0964
12	4.0549	0.2373	271.3629	1.2905	1.8617	0.1094	1.8551	0.0999

Table 4.23: Average and maximal relative error of different heuristic solution approaches for mixed source delays.

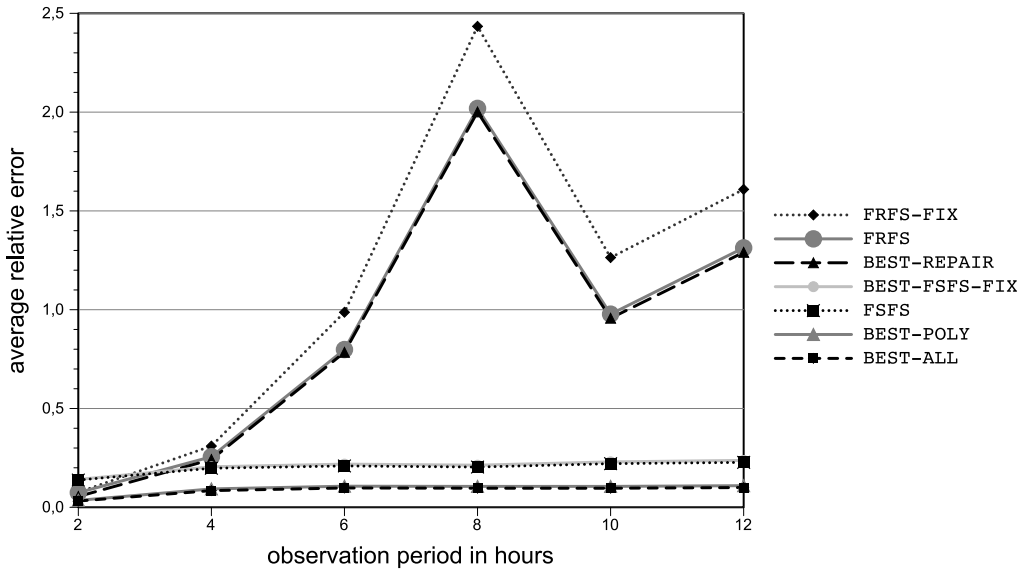

Figure 4.24: Average relative error of different heuristic solution approaches, depending on the size of the observation period, for mixed source delays.

hours	Fsfs	Frfs	Frfs-Fix
2	99.05%	93.81%	92.86%
4	98.10%	90.71%	90.71%
6	97.14%	90.00%	89.52%
8	97.62%	89.76%	89.52%
10	96.67%	91.67%	91.67%
12	96.43%	90.95%	90.24%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	98.10%	93.57%	98.57%	99.05%
4	98.10%	90.71%	98.33%	98.33%
6	96.43%	89.76%	96.90%	97.62%
8	97.38%	89.76%	97.38%	97.62%
10	96.67%	91.67%	96.90%	96.90%
12	95.95%	90.71%	96.43%	96.90%

Table 4.25: In the case of small source delays, how often is the objective value of a heuristic not larger than 105% of the objective value of an optimal solution?

hours	Fsfs	Frfs	Frfs-Fix
2	98.33%	93.10%	92.14%
4	97.86%	90.48%	90.48%
6	94.76%	88.81%	88.33%
8	96.67%	88.81%	87.38%
10	95.48%	90.95%	90.24%
12	94.76%	89.52%	88.81%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	97.38%	92.38%	97.62%	98.33%
4	97.86%	90.48%	98.10%	98.10%
6	93.81%	88.10%	94.05%	95.00%
8	95.71%	88.10%	95.71%	96.67%
10	95.24%	90.95%	95.71%	95.95%
12	94.29%	89.29%	94.76%	95.24%

Table 4.26: In the case of small source delays, how often is the objective value of a heuristic equal to the objective value of an optimal solution?

hours	Fsfs	Frfs	Frfs-Fix
2	38.71%	94.62%	94.62%
4	2.69%	70.70%	64.78%
6	1.35%	55.14%	48.92%
8	0.27%	56.56%	51.37%
10	1.10%	53.97%	46.85%
12	0.82%	57.81%	50.41%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	38.71%	94.62%	94.89%	94.89%
4	2.69%	70.70%	71.77%	72.58%
6	1.35%	54.59%	54.59%	55.41%
8	0.27%	56.83%	57.10%	57.65%
10	0.82%	53.42%	53.70%	55.89%
12	0.82%	57.53%	58.08%	59.18%

Table 4.27: In the case of large source delays, how often is the objective value of a heuristic not larger than 120% of the objective value of an optimal solution?

hours	Fsfs	Frfs	Frfs-Fix
2	13.17%	73.39%	73.12%
4	0.27%	34.95%	34.14%
6	0.00%	27.84%	27.03%
8	0.00%	27.87%	26.78%
10	0.00%	26.30%	24.38%
12	0.00%	27.40%	24.93%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	13.17%	72.85%	74.19%	74.73%
4	0.27%	36.02%	36.29%	37.90%
6	0.00%	28.65%	28.65%	30.00%
8	0.00%	27.60%	27.60%	28.96%
10	0.00%	24.93%	24.93%	27.12%
12	0.00%	27.40%	27.40%	28.22%

Table 4.28: In the case of large source delays, how often is the objective value of a heuristic not larger than 105% of the objective value of an optimal solution?

hours	Fsfs	Frfs	Frfs-Fix
2	47.54%	65.90%	64.26%
4	30.07%	37.58%	36.27%
6	30.17%	31.53%	29.49%
8	30.39%	30.74%	30.04%
10	28.04%	34.46%	32.09%
12	25.00%	34.93%	32.53%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	46.89%	67.87%	77.05%	79.34%
4	28.10%	39.22%	50.65%	54.58%
6	26.44%	33.56%	46.44%	52.20%
8	27.21%	32.86%	44.88%	48.76%
10	25.34%	34.80%	44.93%	50.00%
12	21.58%	33.56%	42.12%	47.95%

Table 4.29: In the case of mixed source delays, how often is the objective value of a heuristic not larger than 105% of the objective value of an optimal solution?

hours	Fsfs	Frfs	Frfs-Fix
2	36.39%	45.90%	45.25%
4	14.05%	23.53%	21.24%
6	16.27%	17.63%	15.93%
8	13.78%	16.61%	15.19%
10	12.84%	20.61%	18.24%
12	12.33%	20.89%	19.52%

hours	Best-Fsfs-Fix	Best-Repair	Best-Poly	Best-All
2	35.41%	44.26%	55.41%	60.66%
4	10.78%	20.26%	24.51%	32.03%
6	12.54%	15.93%	22.71%	30.51%
8	12.37%	18.37%	26.15%	28.98%
10	11.15%	18.92%	25.34%	31.08%
12	9.59%	17.81%	22.26%	28.77%

Table 4.30: In the case of mixed source delays, how often is the objective value of a heuristic not larger than 101% of the objective value of an optimal solution?

4.4 Summary

In Section 4.1 and Section 4.2, we have shown that the relative error of *each* solution approach

- that fixes the order of trains as it is in the original timetable (see Theorem 4.8),
- that fixes the order of trains according to the optimal solution of the corresponding instance of the uncapacitated delay management problem (see Theorem 4.9),
- that applies a no-wait policy (see Theorem 4.13),
- that applies an all-wait policy (see Theorem 4.14), or
- that relaxes the capacity constraints, computes a solution of the relaxed problem, and "repairs" that solution afterwards (see Theorem 4.15)

might get arbitrarily large. Hence the worst-case analysis in this chapter does not only hold for the heuristics presented so far, but for whole classes of heuristics.

As we have shown in Section 4.3, for the real-world data set on which our case study is based, the *average* relative error of the solution procedures suggested in this chapter is rather small, compared to the worst-case error bounds proven in Section 4.1 and Section 4.2. Nevertheless, on few instances, we indeed have really high relative errors of Frfs, Frfs-Fix, and Priority-Repair which is reflected by high maximal relative errors especially in Tables 4.10, 4.12, 4.16, 4.21, and 4.23.

Another important result is that the relative errors of the heuristics suggested in Section 4.1 and Section 4.2 as well as which heuristic is the best one highly depends on the distribution of the source delays:

If only small source delays occur, solution approaches which do not change the order of trains lead to significantly better results than solution approaches which might change the order of trains, and an all-wait strategy yields better solutions than a no-wait strategy. Hence, in the case of small source delays, Fsfs and Best-Fsfs-Fix are a better choice than Frfs, Frfs-Fix, and Best-Repair. As the relative errors of Fsfs and Best-Fsfs-Fix are close together in this case, Best-Fsfs-Fix probably is the best choice as it has guaranteed polynomial runtime – this is important for large-scale real-world instances.

If only large source delays occur, things totally change; in this case, the maximal and average relative errors of heuristics which might change the order of trains are smaller than those of heuristics which fix the order of trains according to the original timetable

(as long as long-term effects are not taken into account). In addition, the number of scenarios in which they are "close" to the optimum is by several orders of magnitude larger. In contrast to the case of small source delays, a no-wait strategy leads to better results than an all-wait strategy if the source delays are rather large. Hence, in the case of large source delays, FRFS, FRFS-FIX, and BEST-REPAIR are superior to FSFS and BEST-FSFS-FIX if mostly short-term effects (up to six hours) are considered. In our case study, BEST-REPAIR is the best choice for large source delays (if the focus lies on short-term effects): it has a rather small average relative error and guaranteed polynomial runtime.

As a consequence of this irregular behavior and due to the observation that in many scenarios in which one heuristics has a rather large relative error, there are other heuristics which have a significantly smaller relative error, it is preferable to use an approach like BEST-POLY or BEST-ALL that combines different heuristics, some which are good for small source delays, some which are good for large source delays. In conclusion, BEST-POLY is a heuristic which is a good trade-off between runtime and accuracy:

- BEST-POLY yields a small average relative error, and even the maximal relative error is rather limited. Its average and maximal relative error is not significantly higher than the relative error of the BEST-ALL heuristic.
- In many cases, the solution computed by BEST-POLY is close to the optimum as we have shown in Tables 4.25-4.30. The number of cases in which the solution of BEST-POLY is close to the optimal solution is almost as large as it is for the BEST-ALL heuristic.
- The runtime of BEST-POLY is linear in the size of the event-activity network, while for BEST-ALL, several NP-hard problems have to be solved. For the instances of our case study, there is no significant difference between the runtimes of both heuristics (for short observation periods, BEST-ALL is even faster than BEST-POLY, while for larger observation periods, BEST-POLY is faster than BEST-ALL, see Table 4.18). For very large instances, due to its guaranteed linear runtime, BEST-POLY is preferable to BEST-ALL.

Chapter 5

Integration with Rolling Stock Circulations

In this chapter, we show how rolling stock circulations can be integrated into delay management. In Section 5.1, we show how this can be modeled in the event-activity network and which changes to the integer programming formulation are necessary. We transfer some results from Chapter 3 to the extended problem in Section 5.2. In Section 5.3, we prove that the extended problem is NP-hard, even in very special cases and even if wait/depart decisions and priority decisions are neglected. In Section 5.4, we present a special case where the problem can be solved in polynomial time, and we suggest a generic solution framework for solving it in other cases in Section 5.5. Some parts of Section 5.1, Section 5.3, and Section 5.4 have been reported in [FNSS07].

5.1 Model

The starting point for integrating rolling stock circulations and delay management is the event-activity network. To model changes of the rolling stock circulations within the delay management stage, we need to specify the set $\mathcal{T}$ of all trips where each trip is given as a path in the network $\mathcal{N}_{\text{train}} = (\mathcal{E}, \mathcal{A}_{\text{train}})$ and represents a train operating a line from its start station to its end station. Each trip $\tau \in \mathcal{T}$ is specified by its start event $s(\tau) \in \mathcal{E}_{\text{start}}$ and its end event $e(\tau) \in \mathcal{E}_{\text{end}}$ with

$$\mathcal{E}_{\text{start}} := \{i \in \mathcal{E}_{\text{dep}} \colon i \text{ is the first event of a trip } \tau \in \mathcal{T}\}$$
$$\mathcal{E}_{\text{end}} := \{i \in \mathcal{E}_{\text{arr}} \colon i \text{ is the last event of a trip } \tau \in \mathcal{T}\}\,.$$

In the vehicle schedule, the trips are combined to rolling stock circulations, i.e. each trip is assigned to a train which serves this trip. Each circulation c consists of a sequence of

trips $(\tau_{c,1}, \tau_{c,2}, \ldots, \tau_{c,n_c})$ where the first event of the circulation is $s(\tau_{c,1})$ and its last event is $e(\tau_{c,n_c})$. Note that each driving and each waiting activity belongs to exactly one trip (and hence to exactly one circulation).

Before explaining the model in more detail, we briefly show why it might make sense to take into account rolling stock circulations during delay management. To this end, assume that we have to trains A and B, let $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$, and assume that trips τ_1 and τ_4 are served by train A, while trips τ_2 and τ_3 are served by train B. Furthermore, assume that trip τ_3 follows shortly after trip τ_2 while train A has a long waiting time between its two trips and that train B gets a delay while serving its first trip τ_2. See Figure 5.1 for an illustration.

If the delay of train B is such large that it cannot start trip τ_3 on time, then it might make sense to swap the assignment of trips τ_3 and τ_4. In this case, train A serves trip τ_3 on time, while train B serves trip τ_4 on time or with only a small delay. A similar problem occurs for example during disruption management in an airline setting where *crew swapping* is a common reaction to delays, see for example [LJN00] and [SK06].

Now, we explain the model in more detail. We assume that before serving its first trip, each train waits in some depot, and after serving its last trip, each train drives back to some depot (we do not make any assumptions on the number and locations of the depots). To model this, we add an "initial arrival event" for each train and set its time in the timetable to the time when the train is ready to leave its depot. Note that although this event models the *departure* from the depot, for technical reasons, it is an *arrival* event as circulation activities always start with an arrival event. At a first glance, this might be confusing, so this event can also be interpreted as the arrival of

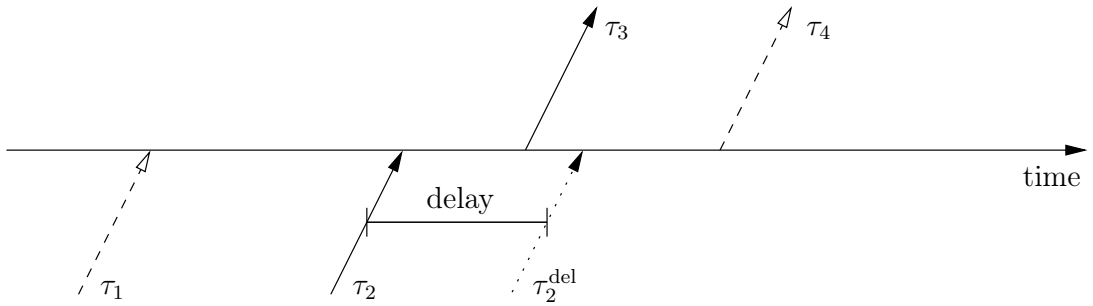

Figure 5.1: An example why changes to the rolling stock circulations during delay management might make sense. The solid arrows represent trips τ_2 and τ_3 assigned to train B, while dashed arrows stand for trips τ_1 and τ_4 served by train A. The dotted arrow represents the delayed trip τ_2.

the train directly after leaving a maintenance area or a track where it has been parked before driving to the station where its first trip starts (and this driving then is modeled by a circulation activity). Similarly, we add a "terminal departure event" for each train that models its driving to its depot. The sets of all those events are denoted by

$$\mathcal{E}_{\text{init}} \quad \text{and} \quad \mathcal{E}_{\text{term}}.$$

Based on the definitions above, all potential combinations between two trips can now be defined and included in the event-activity network. To this end, we introduce a new type of activity, analogously to the types already defined in Section 2.2:

- *Circulation activities* $\mathcal{A}_{\text{circ}} \subset (\mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{init}}) \times (\mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}})$ specify which trips can be operated consecutively by the same train within the same circulation. More specific, if τ_1 is a trip with end event i and τ_2 a trip starting with event j, then we are allowed to operate trip τ_1 and trip τ_2 consecutively by the same train if $a = (i, j) \in \mathcal{A}_{\text{circ}}$.

We distinguish two cases, depending on whether a circulation activity connects two trips or whether it connects a trip to the start depot or to the end depot of a train:

- If the start event i of a circulation activity $a = (i, j)$ is the end event of some trip τ_1 and if the end event j is the start event of another trip τ_2, i.e. if $i \in \mathcal{E}_{\text{end}} \subset \mathcal{E}_{\text{arr}}$ and $j \in \mathcal{E}_{\text{start}} \subset \mathcal{E}_{\text{dep}}$, then the lower bound L_a is set to the waiting time that is needed within the station belonging to event i to let passengers get off the train, to allow a crew change etc., plus the time needed to drive to the station/platform belonging to event j, plus the time needed there for technical reasons and for letting passengers get on the train. Note that a circulation activity between i and j is only possible if the types of the two trains required for trips τ_1 and τ_2 are compatible and if
$$\pi_j \geq \pi_i + L_a. \tag{5.1}$$
- To model a train leaving its depot for its first trip, we add a circulation activity from its initial arrival event $i \in \mathcal{E}_{\text{init}}$ to the first event of each trip that this train could serve. The lower bound L_a of the corresponding circulation activity is set to the time the train needs to drive from its depot to the station where the trip starts plus some waiting time to let the passengers get on the train. Hence, all activities starting with some event $i \in \mathcal{E}_{\text{init}}$ are circulation activities ending with some event $j \in \mathcal{E}_{\text{start}}$. The same holds for $\mathcal{E}_{\text{end}}$ and $\mathcal{E}_{\text{term}}$ – those circulation activities model a train driving to its depot at the end of the day.

In order to obtain feasible circulations, we require that for each event $i \in \mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{init}}$ ($j \in \mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}}$), exactly one circulation activity starting in i (ending in j) has to be

respected. To model this in the IP formulation (DM), we add a new type of decisions to the two types we already used so far (wait/depart and priority): To each trip $\tau \in \mathcal{T}$ with end event $e(\tau) \in \mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{init}}$, assign exactly one trip $\tau' \in \mathcal{T}$ with start event $s(\tau') \in \mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}}$. We call those decisions *circulation decisions*. To add them to the IP formulation of the delay management problem, we introduce binary decision variables

$$v_{ij} := \begin{cases} 1 & \text{if activity } (i,j) \in \mathcal{A}_{\text{circ}} \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

for all circulation activities $(i,j) \in \mathcal{A}_{\text{circ}}$. Then, the integer programming formulation of the delay management problem with integrated rolling stock circulations looks as follows:

$$(\mathbf{DMC}) \quad \min f(x,z,g) = \sum_{i \in \mathcal{E}} w_i (x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a w_a T \tag{5.2}$$

such that

$$\begin{aligned}
x_i &\geq \pi_i + d_i && \forall\, i \in \mathcal{E} && (5.3)\\
x_j - x_i &\geq L_a + d_a && \forall\, a = (i,j) \in \mathcal{A}_{\text{train}} && (5.4)\\
M z_a + x_j - x_i &\geq L_a && \forall\, a = (i,j) \in \mathcal{A}_{\text{change}} && (5.5)\\
M g_{ij} + x_j - x_i &\geq L_{ij} && \forall\, (i,j) \in \mathcal{A}_{\text{head}} && (5.6)\\
g_{ij} + g_{ji} &= 1 && \forall\, (i,j) \in \mathcal{A}_{\text{head}} && (5.7)\\
M(1 - v_{ij}) + x_j - x_i &\geq L_a && \forall\, a = (i,j) \in \mathcal{A}_{\text{circ}} && (5.8)\\
\sum_{i \in (\mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{init}})} v_{ij} &= 1 && \forall\, j \in \mathcal{E}_{\text{start}} && (5.9)\\
\sum_{i \in \mathcal{E}_{\text{end}}} v_{ij} &= 1 && \forall\, j \in \mathcal{E}_{\text{term}} && (5.10)\\
\sum_{j \in (\mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}})} v_{ij} &= 1 && \forall\, i \in \mathcal{E}_{\text{end}} && (5.11)\\
\sum_{j \in \mathcal{E}_{\text{start}}} v_{ij} &= 1 && \forall\, i \in \mathcal{E}_{\text{init}} && (5.12)\\
x_i &\in \mathbb{N} && \forall\, i \in \mathcal{E} && (5.13)\\
z_a &\in \{0,1\} && \forall\, a \in \mathcal{A}_{\text{change}} && (5.14)\\
g_{ij} &\in \{0,1\} && \forall\, (i,j) \in \mathcal{A}_{\text{head}} && (5.15)\\
v_{ij} &\in \{0,1\} && \forall (i,j) \in \mathcal{A}_{\text{circ}} && (5.16)
\end{aligned}$$

where again M is a constant which is "large enough" (see Theorem 5.1). Note that the objective does not change and that constraints (5.3)-(5.7) and (5.13)-(5.15) are the same as constraints (2.2)-(2.9) in the original IP formulation of (DM) – we have included them here for the sake of clarity only.

We shortly explain the meaning of the additional constraints:

- Constraints (5.8) ensure that if activity a is chosen, i.e. if $v_{ij} = 1$, then its lower bound has to be respected. Otherwise, due to M being "large enough", we do not further restrict x.
- Each trip needs exactly one predecessor (constraints (5.9)-(5.10)) and exactly one successor (constraints (5.11)-(5.12)), and no train is allowed to directly drive from its start depot to its end depot.

If the circulation activities that are used when operating the original timetable π are all included in $\mathcal{A}_{\text{circ}}$, then (DMC) is feasible; one feasible solution for example can be computed by fixing wait/depart decisions, priority decisions, and circulation decisions as they are in the original timetable and solving the resulting instance of (PP($\mathcal{A}_{\text{fix}}$)).

Since (DM) is NP-hard as already mentioned in the beginning of Chapter 4, (DMC) as a generalization of (DM) is NP-hard, too. Note that the additional constraints (5.9)-(5.12) and (5.16) form a matching problem. Although "pure" matching problems can be solved in polynomial time (see for example [BDM09]), we show in Theorem 5.3 and Theorem 5.4 that even if $\mathcal{A}_{\text{change}} = \mathcal{A}_{\text{head}} = \emptyset$, the matching problem in the context of delay management is NP-hard even in very simple special cases. However, we first show that most of the results from Chapter 3 can be extended to the delay management problem with integrated rolling stock circulations.

5.2 Analyzing the Model

Due to the fact that we only take into account circulation activities satisfying (5.1), Theorem 3.1 and Corollary 3.2 still hold for (DMC):

Theorem 5.1. *Given an instance of (DMC),*

$$M = \max_{i \in \mathcal{E}} d_i + \sum_{a \in \mathcal{A}_{\text{train}}} d_a + \sum_{(i,j) \in \mathcal{A}_{\text{head}}^{\text{back}}} \pi_i - \pi_j + L_{ij}$$

is "large enough".

Proof. The proof of Theorem 3.1 only needs some minor modifications: For a solution (x, z, g, v) of (DMC), we denote the set of fixed circulation activities in this solution by

$$\mathcal{A}^{\text{fix}}_{\text{circ}} := \{(i,j) \in \mathcal{A}_{\text{circ}} : v_{ij} = 1\}.$$

To take into account those fixed circulation activities when applying the Critical Path Method (Algorithm 2.1) for computing the disposition timetable $\tilde{x}$, we define

$$\mathcal{A}_{\text{fix}} := \mathcal{A}^{\text{fix}}_{\text{change}} \cup \mathcal{A}^{\text{fix}}_{\text{head}} \cup \mathcal{A}^{\text{fix}}_{\text{circ}}.$$

Parts 1), 2) a), and 2) b) of the proof keep the same, and for the case that in part 2), the activity $a = (i,k)$ that carries over the delay is a circulation activity, we add part 2) c) as follows:

c) If $a = (i,k) \in \mathcal{A}^{\text{fix}}_{\text{circ}}$, then

$$\begin{aligned}\tilde{x}_k &= \tilde{x}_i + L_a \\ &\leq \pi_i + U_i + L_a \\ &\leq \pi_k - L_a + U_i + L_a \\ &= \pi_k + U_i \\ &\leq \pi_k + U_k\end{aligned}$$

where the second line is a direct consequence of the induction hypothesis for event $i \prec k$ and the third line is due to the fact that we only allow circulation activities satisfying (5.1).

This completes the proof of Theorem 3.1. To extend the proof of Corollary 3.2 to the delay management problem with integrated rolling stock circulations, it remains to show that in an optimal solution, for any $(i,j) \in \mathcal{A}_{\text{circ}}$ with $v_{ij} = 0$, (5.8) is satisfied "automatically". From the extension of Theorem 3.1 to delay management with integrated rolling stock circulations that we just have shown, we know that $x_i \leq \pi_i + M$, and as x satisfies constraints (5.3), we have $x_j \geq \pi_j$. As we require (5.1) to be satisfied by each circulation activity, we have $\pi_j - \pi_i \geq L_a$, hence

$$\begin{aligned}M(1 - v_{ij}) + x_j - x_i &= M + x_j - x_i \\ &\geq M + \pi_j - (\pi_i + M) \\ &= \pi_j - \pi_i \\ &\geq L_a.\end{aligned}$$

□

If we do not limit the set of possible circulation activities by (5.1), but allow circulation activities between all trips served by compatible trains, Theorem 5.1 still holds if we

take into account the additional delay caused by fixing a circulation activity $a = (i, j)$ with $\pi_j < \pi_i + L_a$, i.e. if we choose

$$M = \max_{i \in \mathcal{E}} d_i + \sum_{a \in \mathcal{A}_{\text{train}}} d_a + \sum_{(i,j) \in \mathcal{A}_{\text{head}}^{\text{back}}} (\pi_i - \pi_j + L_{ij}) + \sum_{\substack{a=(i,j) \in \mathcal{A}_{\text{circ}}: \\ \pi_j < \pi_i + L_a}} (\pi_i - \pi_j + L_a).$$

As the proof is rather technical and very similar to the proofs of Theorem 3.1, Corollary 3.2, and Theorem 5.1, we omit it here.

If we bound the delay of each event analogously to problem (BDM) presented in Section 3.2, we obtain the delay management problem with integrated rolling stock circulations and bounded delay (BDMC):

$$(\textbf{BDMC}) \ \min f(x, z, g) = \sum_{i \in \mathcal{E}} w_i (x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a w_a T$$

such that

$$x_i \leq \pi_i + Y \quad \forall i \in \mathcal{E}$$

and such that (5.3)-(5.16) are satisfied.

As in Section 3.2, we can use this restriction to give a tight bound on the constant M and to yield a reduction approach for reducing the size of the input instance. Theorem 3.5 still holds for (BDMC):

Theorem 5.2. *Given an instance of (BDMC),*

$$M = Y + \max_{(i,j) \in \mathcal{A}_{\text{head}}} (\pi_j - \pi_i + L_{ji})$$

is "large enough".

Proof. The proof is similar to the proof of Theorem 3.5. It remains to show that for each $a = (i, j) \in \mathcal{A}_{\text{circ}}$ with $v_{ij} = 0$, constraint (5.8) is fulfilled "automatically" in an optimal solution:

$$\begin{aligned} M(1 - v_{ij}) + x_j - x_i &= M + x_j - x_i \\ &= Y + \max_{(i,j) \in \mathcal{A}_{\text{head}}} (\pi_j - \pi_i + L_{ji}) + x_j - x_i \\ &\geq Y + x_j - x_i \\ &\geq Y + \pi_j - (\pi_i + Y) \\ &= \pi_j - \pi_i \\ &\geq L_a \end{aligned}$$

where the last inequality is due to the fact that we allow circulation activities only between two events i and j if (5.1) is satisfied – with this restriction, a circulation activity can only carry over an existing delay, but not increase it (in a time-minimal solution, we have $x_j - \pi_j = x_i + L_a - \pi_j \leq x_i - \pi_i$ if (5.1) is fulfilled). However, if we drop assumption (5.1) and allow circulation activities between all trips served by compatible trains, then Theorem 5.2 still holds if

$$M = Y + \max_{(i,j)\in\mathcal{A}_{\text{head}}} (\pi_j - \pi_i + L_{ji}) + \max_{a=(i,j)\in\mathcal{A}_{\text{circ}}} (\pi_i - \pi_j + L_a).$$

Analogously to the proof above, in this case we have

$$\begin{aligned} M(1 - v_{ij}) + x_j - x_i &\geq \pi_j - \pi_i + \max_{a=(i,j)\in\mathcal{A}_{\text{circ}}} (\pi_i - \pi_j + L_a) \\ &\geq L_a. \end{aligned}$$

□

Furthermore, Theorem 3.6 and the consequences which yield algorithm Fix-Headways still hold, so algorithm Fix-Headways also can be applied to an instance of (BDMC). In addition, the reduction techniques suggested in Section 3.3 are valid for problem (DMC), too, if we apply the following two slight modifications to algorithm Reduce (see page 38).

First, we have to consider the circulation activities in $\mathcal{A}_\pi$, i.e. we have to modify (3.4) in the following way:

$$\mathcal{A}_\pi := \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}^{\text{forw}}_{\text{head}} \cup \mathcal{A}_{\text{circ}}. \tag{5.17}$$

This reflects the fact that each circulation activity could carry over a delay. The modified set $\mathcal{A}_\pi$ has to be used when $\mathcal{E}_{\text{mark}}$ is computed according to equation (3.5).

Secondly, we have to apply a rather technical change in the definition of $\mathcal{E}_{\text{reduced}}$ in Step 2 of algorithm Reduce which we explain using a small example, see Figure 5.2.

In that example, event 1 is source delayed, hence $\mathcal{E}_{\text{mark}} = \{1, 2, 3, 5, 6\}$. Event 4 is not contained in $\mathcal{E}_{\text{mark}}$, hence it is not contained in $\mathcal{E}_{\text{reduced}}$, so in Step 3 of algorithm Reduce, the circulation activities $(4, 2)$ and $(4, 5)$ are not contained in $\mathcal{A}_{\text{reduced}}$, yielding an infeasible problem. Thus we have to modify $\mathcal{E}_{\text{reduced}}$ in Step 2 of algorithm Reduce as follows:

$$\begin{aligned} \mathcal{E}_{\text{reduced}} := \mathcal{E}_{\text{mark}} &\cup \{i \in \mathcal{E} : \exists a = (i, j) \in \mathcal{A}_{\text{train}} \text{ with } d_a > 0\} \\ &\cup \{i \in \mathcal{E} : \exists (i, j) \in \mathcal{A}_{\text{circ}} \text{ with } j \in \mathcal{E}_{\text{mark}}\}. \end{aligned} \tag{5.18}$$

With these modifications, algorithm Reduce looks as follows:

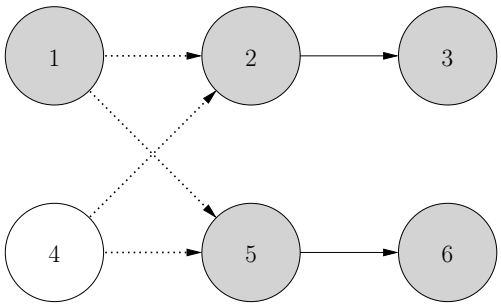

Figure 5.2: If $\mathcal{E}_{\text{reduced}}$ in algorithm REDUCE is not adapted, too many activities are deleted. Solid arrows represent driving activities, dotted arrows are circulation activities. In grey: events contained in $\mathcal{E}_{\text{mark}}$ if event 1 is source delayed.

Algorithm 5.1: Algorithm REDUCE for problem (DMC)

Input: The event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, a timetable π, and source delays $d \in \mathbb{N}^{|\mathcal{E}|+|\mathcal{A}_{\text{train}}|}$.

Step 1: Calculate $\mathcal{E}_{\text{mark}}$ according to (3.5), using the definition of $\mathcal{A}_\pi$ from (5.17).

Step 2: Compute $\mathcal{E}_{\text{reduced}}$ according to (5.18).

Step 3: Compute $\mathcal{A}_{\text{reduced}} := \{(i,j) \in \mathcal{A} : i, j \in \mathcal{E}_{\text{reduced}}\}$.

Output: The reduced event-activity network $\mathcal{N}_{\text{reduced}} = (\mathcal{E}_{\text{reduced}}, \mathcal{A}_{\text{reduced}})$.

This modification of algorithm REDUCE (and hence also the corresponding modification of algorithm FIX-AND-REDUCE) can also be applied to an instance of (BDMC) as Theorem 3.8, Theorem 3.9, and Theorem 3.10 are still valid as long as (5.1) is satisfied (as we have shown in the proof of Theorem 5.2, in this case, a circulation activity can only carry over an existing delay, but not cause an additional delay). For the proof of Theorem 3.8, the circulation activities can be treated like the other activities in $\mathcal{A}_\pi$.

After solving the problem on the reduced network computed by algorithm REDUCE, for all $(i,j) \in \mathcal{A}_{\text{circ}} \setminus \mathcal{A}_{\text{reduced}}$, we have to set $v_{ij} = 1$ if (i,j) is chosen in the original plan, 0 otherwise.

Finally, as long as (5.1) is satisfied, by defining $\mathcal{A}_\pi$ as in (5.17), we can also transfer the results from Section 3.4 to the delay management problem with integrated rolling stock

circulations, especially the definition of the never-meet property: Definition 3.12 does not change (except for the fact that the underlying set $\mathcal{A}_\pi$ has changed), Lemma 3.13 still holds as it depends only on Theorem 3.10 and Theorem 3.8 (which, as already mentioned above, also are valid for the extended model), Corollary 3.14 is a direct consequence of Lemma 3.13, and Theorem 3.15 which depends on Lemma 3.13 and Corollary 3.14 is still valid, too.

5.3 Computational Complexity

Now, we show that even very special cases of (DMC) already are NP-hard. To show the first of these results, we need the *Minimum Satisfiability Problem* (MIN k-SAT):

(MIN k-SAT) Given *boolean variables* $X = \{x_1, \ldots, x_n\}$, *clauses* $\mathcal{C} = \{C_1, \ldots, C_m\}$ of the form $C_i = l_{i_1} \vee \ldots \vee l_{i_j}$ $(j \leq k)$ where each *literal* l_i is either the variable x_i or its negation $\bar{x}_i$, and weights $w_1, \ldots, w_m$ for the clauses, find a *truth assignment* $X \to \{\text{true}, \text{false}\}^n$ such that the total weight of the satisfied clauses is minimal.

As was shown in [KKM94], MIN k-SAT is NP-hard for $k \geq 2$.

By reducing MIN 3-SAT to an instance of (DMC) where circulation decisions are possible only in one single station and where $\mathcal{A}_{\text{change}} = \mathcal{A}_{\text{head}} = \emptyset$, we show that even this very simple version of (DMC) is NP-hard:

Theorem 5.3. *(DMC) is NP-hard even in the special case that only pairwise swapping of trips is allowed only in one special station and that no wait-depart decision and no priority decision has to be made.*

Proof. Given an instance $(X, \mathcal{C})$ of MIN 3-SAT with uniform weights, we first show how to construct a corresponding instance of (DMC). For each variable $x_i \in X$, we create four events in the event-activity network: two arrival events $\text{arr}_i^{\text{true}}$ and $\text{arr}_i^{\text{false}}$ and two departure events x_i and $\bar{x}_i$. Then, we connect each of the two arrival events with each of the two departure events by a circulation activity, hence we add the four circulation activities $(\text{arr}_i^{\text{true}}, x_i)$, $(\text{arr}_i^{\text{true}}, \bar{x}_i)$, $(\text{arr}_i^{\text{false}}, x_i)$, and $(\text{arr}_i^{\text{false}}, \bar{x}_i)$ to the event-activity network, see Figure 5.3 for an illustration.

Thus, for each variable, there are two possibilities for fixing the corresponding circulations: either fix

$$\text{Circ}_{\text{true}}(i) := \left\{(\text{arr}_i^{\text{true}}, x_i), (\text{arr}_i^{\text{false}}, \bar{x}_i)\right\},$$

or fix

$$\text{Circ}_{\text{false}}(i) := \left\{(\text{arr}_i^{\text{false}}, x_i), (\text{arr}_i^{\text{true}}, \bar{x}_i)\right\}.$$

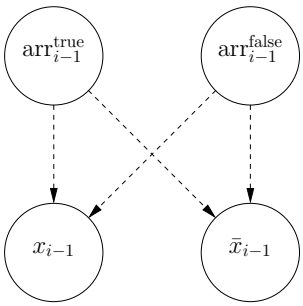

Figure 5.3: For each variable $x_i \in X$, we add two arrival and two departure events, connected by circulation activities, to the event-activity network. The dashed arrows are circulation activities.

For each clause $C_i \in \mathcal{C}$, we add a station with a single arrival event C_i. For each literal l that is part of some clause C_i, we add a driving activity connecting the departure event l (which is either x or $\bar{x}$) and the arrival event C_i. The lower bounds of all activities are set to 1 and we assume a tight timetable without slack times. The source delay of all events $\text{arr}_i^{\text{true}}$ is set to 1. The weights w_i of all events are set to 0, only the weights of the events C_i corresponding to the clauses of MIN 3-SAT are set to 1. See Figure 5.4 for the resulting event-activity network.

Note that the sum of the delays of all events corresponding to a variable $x_i \in X$ in a time-minimal solution of (DMC) is always fixed, as well as their contribution to the objective of (DMC): the arrival event $\text{arr}_i^{\text{true}}$ always has a delay of 1 and the arrival event $\text{arr}_i^{\text{false}}$ always is on time. Depending on the circulation decisions, either event x_i has a delay of 1 while event $\bar{x}_i$ is on time or vice versa. Furthermore, all those events have a contribution of 0 to the objective, independently of the circulation decisions (as their weights all are equal to 0). The only effect that different circulation decisions have on the objective is how many events C_i get a secondary delay and how many of them are on time: if at least one predecessor of an event C_i has a delay of 1, then C_i also has a delay of 1, while C_i is on time if all of its predecessors are on time. So minimizing the sum of all delays is equivalent to minimizing the number of delayed events C_i.

Using this construction, there exists a truth assignment with k satisfied clauses if and only if there exists a solution of the corresponding instance of (DMC) with objective value k. The relationship between the variables of MIN 3-SAT and the circulation decisions in (DMC) is the following:

$$x_i = \begin{cases} \text{true} & \text{if and only if } \text{Circ}_{\text{true}}(i) \text{ is chosen} \\ \text{false} & \text{if and only if } \text{Circ}_{\text{false}}(i) \text{ is chosen.} \end{cases} \tag{5.19}$$

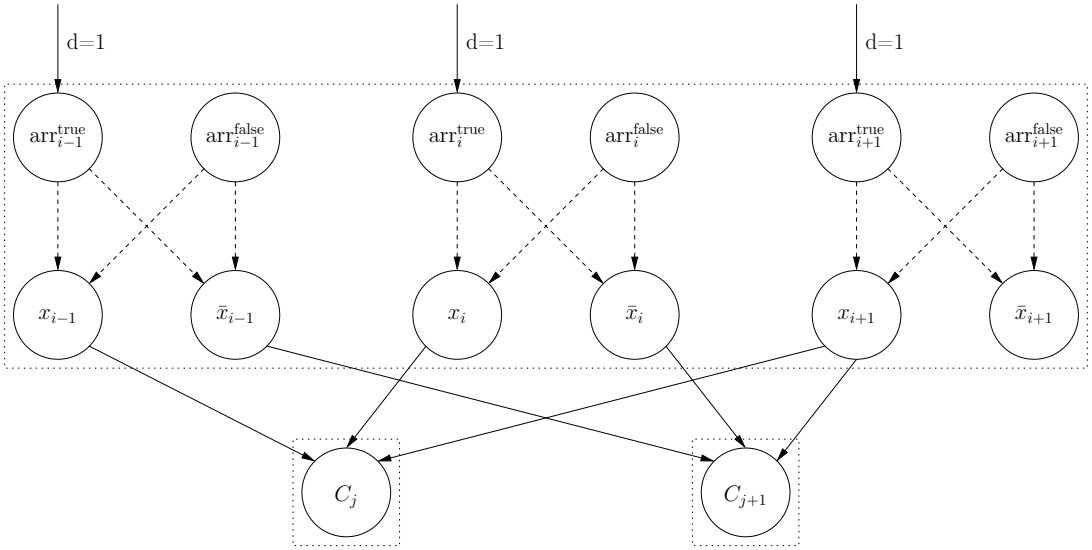

Figure 5.4: The event-activity network constructed from the instance $(X, \mathcal{C})$ of MIN 3-SAT in the proof of Theorem 5.3; in this example, we have $C_j = x_{i-1} \vee x_i \vee x_{i+1}$ and $C_{j+1} = \bar{x}_{i-1} \vee \bar{x}_i \vee x_{i+1}$. Dotted rectangles represent stations, solid arrows are driving activities, dashed arrows are circulation activities.

Due to this relationship, the number of satisfied clauses in MIN 3-SAT is the same as the number of delayed events C_i in (DMC) which coincides with the objective value:

1. Given a solution of MIN 3-SAT, if a clause C_i is satisfied, then there exists at least one literal l_k with $l_k = \text{true}$. W.l.o.g., let $l_k \equiv x_k$ for some $x_k \in X$ (the case $l_k \equiv \bar{x}_k$ works analogously). Then $x_k = \text{true}$, hence by the construction above, the circulation activities in $\text{Circ}_{\text{true}}(k)$ are chosen, thus the delay from event $\text{arr}_k^{\text{true}}$ is carried over to event x_k and further to event C_i – hence event C_i is delayed. If a clause C_i is not fulfilled, then all of its literals are set to false, hence (by the same arguments as above) we fix the circulation activities in such a way that no predecessor of event C_i is delayed, so C_i is on time. As a consequence, by using (5.19), for each truth assignment with k satisfied clauses, we can construct a solution of (DMC) with objective value k.

2. Now, given an optimal solution of (DMC), if an event C_i is delayed, then this delay is carried over from at least one event l_k. W.l.o.g., assume that this event is

x_k (again the case $l_k \equiv \bar{x}_k$ works analogously). Hence x_k has a delay that – in an optimal solution – only can be carried over from event $\text{arr}_k^{\text{true}}$ if the circulation activities in $\text{Circ}_{\text{true}}(k)$ are fixed. Thus we set $x_k = \text{true}$, hence clause C_i is fulfilled. Analogously, if an event C_i is on time, all of its predecessors have to be on time, and by the same arguments as above, the literals in C_i are set in such a way that clause C_i is not fulfilled. Thus, for each solution of (DMC) with objective value k, by using (5.19), we can construct a solution of MIN 3-SAT with k satisfied clauses. □

For the next reduction, we use the *Set-Partition Problem*:

(Set-Partition Problem) Given a set $S = \{s_1, \ldots, s_k\} \subset \mathbb{N}$, decide if S can be partitioned into two sets S_1 and $S_2 = S \setminus S_1$ such that

$$\sum_{s_i \in S_1} s_i = \sum_{s_i \in S_2} s_i.$$

It is well known that the Set-Partition Problem is NP-complete (see [GJ79]).

Now, we can prove the following result:

Theorem 5.4. *(DMC) is NP-hard even if no wait-depart decision and no priority decision has to be made and if we only have two trains which might be (repeatedly) swapped at several consecutive stations.*

Proof. Given the set S, we first show how to construct an instance of (DMC). For each $s_i \in S$, we construct a station with two arrival events $\text{arr}_i^{\text{upper}}$ and $\text{arr}_i^{\text{lower}}$ and two departure events $\text{dep}_i^{\text{upper}}$ and $\text{dep}_i^{\text{lower}}$. The stations are linked by driving activities $(\text{dep}_i^{\text{upper}}, \text{arr}_{i+1}^{\text{upper}})$ and $(\text{dep}_i^{\text{lower}}, \text{arr}_{i+1}^{\text{lower}})$ on which we allow no slack time. In each station, we allow all (four) possible circulation activities and define

$$\text{Circ}_{\text{red}}(i) := \left\{(\text{arr}_i^{\text{upper}}, \text{dep}_i^{\text{upper}}), (\text{arr}_i^{\text{lower}}, \text{dep}_i^{\text{upper}})\right\}$$
$$\text{Circ}_{\text{zero}}(i) := \left\{(\text{arr}_i^{\text{upper}}, \text{dep}_i^{\text{lower}}), (\text{arr}_i^{\text{lower}}, \text{dep}_i^{\text{lower}})\right\}.$$

We assign a slack time of s_i to the activities in $\text{Circ}_{\text{red}}(i)$ and a slack time of zero to the activities in $\text{Circ}_{\text{zero}}(i)$. Hence, we constructed two trips between station i and station $i+1$. We assume that we have two trains A and B which have to operate these trips. If train A operates the "upper" trip between station i and station $i+1$, then train B has to operate the "lower" trip and vice versa. In each station, we can swap this order.

Due to our construction, we have exactly two feasible assignments of trains to trips in each station i, each of them including exactly one circulation activity of $\mathrm{Circ}_{\mathrm{red}}(i)$. This means that exactly one of the two trains can reduce its delay by s_i at station i: Assume that train A arrives with a delay of d_A^i and B arrives with a delay of d_B^i and let $d_A^i, d_B^i \geq s_i$. Then the two possibilities are:

1. Train A takes a circulation activity from the set $\mathrm{Circ}_{\mathrm{red}}(i)$ and train B one from the set $\mathrm{Circ}_{\mathrm{zero}}(i)$. At its departure from station i, the delay of train A is reduced to $d_A^i - s_i$, while the delay of train B has not changed.

2. Vice versa, train A does not reduce its delay, but the delay of train B is reduced to $d_B^i - s_i$.

To finish the construction, we add two more stations: a station 0 with two departure events $\mathrm{dep}_0^{\mathrm{upper}}$ and $\mathrm{dep}_0^{\mathrm{lower}}$ which we connect to the rest of the event-activity network by two driving activities $(\mathrm{dep}_0^{\mathrm{upper}}, \mathrm{arr}_1^{\mathrm{upper}})$ and $(\mathrm{dep}_0^{\mathrm{lower}}, \mathrm{arr}_1^{\mathrm{lower}})$ with zero slack times, and a station $k+1$ with two arrival events $\mathrm{arr}_{k+1}^{\mathrm{upper}}$ and $\mathrm{arr}_{k+1}^{\mathrm{lower}}$ which we connect to the rest of the event-activity network by two driving activities $(\mathrm{dep}_k^{\mathrm{upper}}, \mathrm{arr}_{k+1}^{\mathrm{upper}})$ and $(\mathrm{dep}_k^{\mathrm{lower}}, \mathrm{arr}_{k+1}^{\mathrm{lower}})$ with slack times of $\frac{G}{2}$ where

$$G := \sum_{s_i \in S} s_i.$$

The resulting event-activity network is depicted in Figure 5.5.

We now add source delays of G to each of the two first departure events $\mathrm{dep}_0^{\mathrm{upper}}$ and $\mathrm{dep}_0^{\mathrm{lower}}$, all other source delays are zero. We finally set zero weights for all events, but the two last arrival events $\mathrm{arr}_{k+1}^{\mathrm{upper}}$ and $\mathrm{arr}_{k+1}^{\mathrm{lower}}$ get a weight of 1. The objective of the delay management problem hence is to minimize the sum of the delays of the two events $\mathrm{arr}_{k+1}^{\mathrm{upper}}$ and $\mathrm{arr}_{k+1}^{\mathrm{lower}}$.

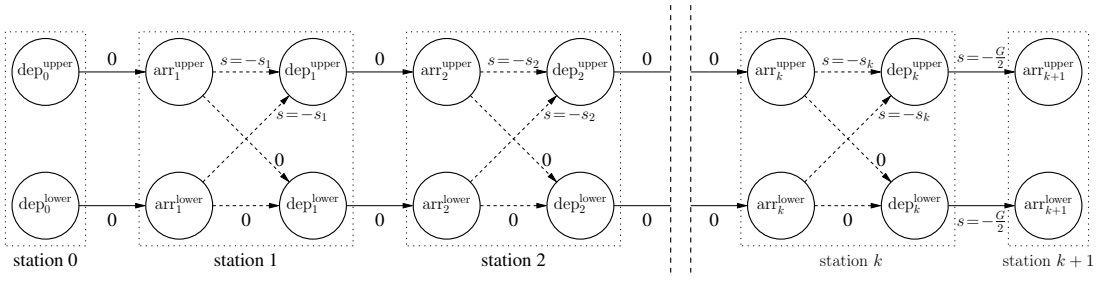

Figure 5.5: The event-activity network constructed from the set S of the Set-Partition Problem in the proof of Theorem 5.4. Dotted rectangles represent stations, solid arrows are driving activities, dashed arrows are circulation activities.

Claim: The Set-Partition Problem has a solution if and only if the objective of the delay management problem is 0.

1. Let (S_1, S_2) be a solution of the Set-Partition Problem. To construct a solution of (DMC), consider each station i:

   - If $s_i \in S_1$, we let train A take a circulation activity from the set $\mathrm{Circ}_{\mathrm{red}}(i)$ and train B one from the set $\mathrm{Circ}_{\mathrm{zero}}(i)$.
   - Otherwise $s_i \in S_2$ and A takes the circulation activity with zero slack while B takes the circulation activity with slack s_i.

   Hence, at the departure from station k, both trains have a delay of

   $$G - \sum_{s_i \in S_1} s_i = G - \sum_{s_i \in S_2} = \frac{G}{2}$$

   since (S_1, S_2) is a solution of the Set-Partition Problem. The last driving activity with slack time $G/2$ finally reduces the delay of both trains to 0, yielding an objective value of 0.

2. Assume that both trains arrive with a delay equal to zero at the last station $k+1$. Then the delays of the departure events $\mathrm{dep}_k^{\mathrm{upper}}$ and $\mathrm{dep}_k^{\mathrm{lower}}$ were smaller than or equal to $G/2$. Since the sum of all slacks is G, both trains A and B reduced their source delays of G by *exactly* $G/2$ until their departure at station k. This means that

   $$S_1 := \{s_i : \text{ train A uses a circulation activity from } \mathrm{Circ}_{\mathrm{red}}(i) \text{ in station } i\}$$
   $$S_2 := \{s_i : \text{ train B uses a circulation activity from } \mathrm{Circ}_{\mathrm{red}}(i) \text{ in station } i\}$$

   is a solution of the Set-Partition Problem. □

5.4 A Polynomially Solvable Case

In the following, we show that a special case of (DMC) can be solved in polynomial time.

Lemma 5.5. *Let $\mathcal{A}_{\mathrm{change}} = \mathcal{A}_{\mathrm{head}} = \emptyset$ and let the sets of successors of all events in $\mathcal{E}_{\mathrm{start}}$ be pairwise disjoint, i.e.*

$$\mathrm{suc}(i) \cap \mathrm{suc}(j) = \emptyset \quad \forall i, j \in \mathcal{E}_{\mathrm{start}}, i \neq j. \tag{5.20}$$

Then (DMC) is equivalent to the problem of computing a minimum cost perfect matching (also known as minimum weighted bipartite matching) in a bipartite graph.

Proof. Given an instance of (DMC) that satisfies the assumptions from Lemma 5.5, we show how to construct the corresponding matching problem. For each $a = (i, j) \in \mathcal{A}_{\text{circ}}$, we have $i \in \mathcal{E}_{\text{end}}$ and $j \in \mathcal{E}_{\text{start}}$, and since $\mathcal{E}_{\text{end}} \cap \mathcal{E}_{\text{start}} = \emptyset$ and due to constraint (5.20), $(\mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{start}}, \mathcal{A}_{\text{circ}})$ is a bipartite graph. We now show how to define the weight function D for the matching problem:

1. Using algorithm CPM, we compute x_k for all $k \in \mathcal{E}_{\text{end}}$.
2. For each $a = (i, j) \in \mathcal{A}_{\text{circ}}$:
   a) Set $x_j^{(a)} := x_i + L_a$.
   b) Use algorithm CPM to compute $x_k^{(a)}$ for all $k \in \text{suc}(j)$.
   c) Set $D(a) := \sum_{k \in \text{suc}(j)} w_k(x_k^{(a)} - \pi_k)$.

As the event-activity network is cycle-free and as no wait/depart decisions and no priority decisions have to be made, all circulation decisions only affect activities $j \in \mathcal{E}_{\text{start}}$ and their subsequent events and no events in $\mathcal{E}_{\text{end}}$ due to constraint (5.20). Hence the first step can be done in time $\mathcal{O}(|\mathcal{E}|)$. For the second step, we need at most time $\mathcal{O}(|\mathcal{A}_{\text{circ}}| \cdot |\mathcal{E}|)$, hence D can be computed in time $\mathcal{O}(|\mathcal{A}_{\text{circ}}| \cdot |\mathcal{E}|)$.

Now, let

$$\mathcal{E}_1 := \bigcup_{k \in \mathcal{E}_{\text{end}}} \text{suc}(k)$$

denote the set of all end events of trips and their successors. Then there exists a solution of the minimum cost perfect matching problem in the bipartite graph $(\mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{start}}, \mathcal{A}_{\text{circ}})$ with cost function D with objective value K if and only if there exists a solution of (DMC) with objective value $K + \sum_{k \in \mathcal{E}_1} w_k(x_k - \pi_k)$:

Given a perfect matching $M \subseteq \mathcal{A}_{\text{circ}}$ with weight K, we define

$$v_{ij} := \begin{cases} 1 & \text{if } (i, j) \in M \\ 0 & \text{otherwise} \end{cases}$$

and compute x_k for all $k \in \mathcal{E}$ by applying algorithm CPM. As M is a perfect matching, v as defined above satisfies (5.9)-(5.12) and (5.16), and by construction of D and due to (5.20), the objective value of (DMC) is

$$\begin{aligned} f(x, z, g, v) &= \sum_{i \in \mathcal{E}_1} w_i(x_i - \pi_i) + \sum_{i \in \mathcal{E} \setminus \mathcal{E}_1} w_i(x_i - \pi_i) \\ &= \sum_{i \in \mathcal{E}_1} w_i(x_i - \pi_i) + \sum_{a=(i,j) \in \mathcal{A}_{\text{circ}}} v_{ij} D(a) \\ &= \sum_{i \in \mathcal{E}_1} w_i(x_i - \pi_i) + K. \end{aligned}$$

Similarly, given a solution of (DMC) with objective value $K + \sum_{k \in \mathcal{E}_1} w_k(x_k - \pi_k)$, $M := \{a = (i,j) \in \mathcal{A}_{\text{circ}} : v_{ij} = 1\}$ is a perfect matching with objective value K. □

Note that a minimum cost perfect matching can be computed in polynomial time, for example in time $\mathcal{O}(n^3)$ where $n = |\mathcal{E}_{\text{end}} \cup \mathcal{E}_{\text{start}}|$ by using the *Hungarian method* (also known as *Kuhn-Munkres algorithm*), see [Kuh55] for its original $\mathcal{O}(n^4)$ formulation or [BDM09] for an overview of available solution approaches.

5.5 Solution Approaches

As we have shown in Section 5.3, (DMC) in general is NP-hard. One approach to nevertheless solve the problem, even if (5.20) is not fulfilled, is to solve the problem on the reduced event-activity network computed by algorithm Fix-And-Reduce; for this approach, we have shown rather good results (for delay management without rolling stock circulations) in our case study in Section 3.5. However, even after a reduction of the input instance, it might still take too long to solve the remaining problem to optimality. Hence, in this section, we suggest a generic solution framework which in general does not compute an optimal, but at least a feasible solution.

First, we present the generic framework; the main idea is to solve the delay management problem with fixed circulation decisions and to look for an improvement of the circulation decisions alternately. After presenting the generic framework, we discuss different possible approaches for each step.

Algorithm 5.2: Local-Improvement

Step 1: Fix the circulation activities.

Step 2: Solve the corresponding instance of (DM) with fixed circulation activities.

Step 3: Find a local improvement of the circulation activities (i.e. one that allows to decrease the disposition time of at least one event $i \in \mathcal{E}_{\text{start}}$). If none found: end.

Step 4: Go to Step 2.

For each of the three first steps of algorithm Local-Improvement, we roughly discuss some ideas:

In Step 1, an initial assignment for the circulation decisions can for example be given by fixing them as they are in the original plan or by using the following greedy approach:

Fix all connections and all headways according to the original timetable, then run algorithm CPM. Whenever the actual event i has outgoing circulation activities, fix that outgoing circulation activity a for which the scheduled time of its end event j is minimal (or for which $x_i + L_a$ is minimal), provided that j has no fixed incoming circulation activity yet. Note that depending on the circulation activities, this approach might not find a feasible solution even if one exists. To demonstrate this, we consider the event-activity network depicted in Figure 5.6.

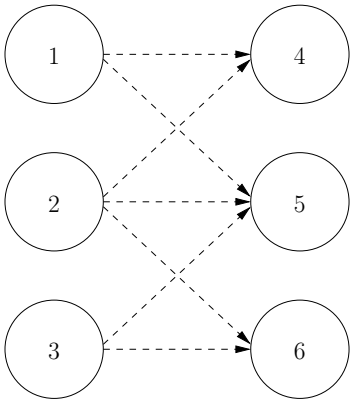

Figure 5.6: Depending on the set of available circulation activities, the greedy approach might not find a feasible assignment of trips to trains, although one exists. Dotted arrows represent circulation activities.

If the first circulation activity that the greedy approach fixes in this example is circulation activity $(1, 5)$ and if $(2, 6)$ is the next one, then the trip ending with event 3 and the trip starting with event 4 cannot be connected to any other trip – although a feasible solution exists.

Once the circulation decisions are fixed, the solution of the corresponding instance of (DM) in Step 2 of algorithm LOCAL-IMPROVEMENT can be computed exactly (either on the original event-activity network or on the reduced event-activity network computed by one of the reduction techniques suggested in Chapter 3) or by applying one of the heuristic solution approaches suggested in Chapter 4.

As we have shown in Section 5.3, computing optimal circulation decisions in general is NP-hard (as a circulation decision might depend on other wait/depart, priority, or circulation decisions – if all circulation decisions are independent of other decisions, fixing the circulation activities reduces to a polynomially solveable matching problem, see Section 5.4). Hence, in Step 3 of algorithm LOCAL-IMPROVEMENT, we are not looking for optimal circulation decisions, but for an improvement of the current situation.

For finding such an improvement, one could pairwise swap the assignment of two trips, search for a local optimum in one single station (i.e. exactly solve the matching problem in that station), or use the approach described in the following lemma:

Lemma 5.6. *Let (x, z, g, v) be a feasible solution of (DMC) where x is a time-minimal timetable, and let $\tilde{v}$ be another feasible assignment of the circulation decisions that fulfills* (5.9)-(5.12) *and* (5.16)*. For fixed $k \in \mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}}$, let $i(k)$, $\tilde{i}(k)$ denote those events for which $v_{i(k)k} = \tilde{v}_{\tilde{i}(k)k} = 1$, i.e. the start events of both circulation activities ending in k that have been fixed by v or by $\tilde{v}$, respectively. If*

$$x_{\tilde{i}(k)} + L_{\tilde{i}(k)k} \leq x_{i(k)} + L_{i(k)k} \quad \forall\, k \in \mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{term}}, \tag{5.21}$$

then there exists a solution $(\tilde{x}, z, g, \tilde{v})$ of (DMC) with $\tilde{x}_j \leq x_j \forall j \in \mathcal{E}$ and

$$f(\tilde{x}, z, g, \tilde{v}) \leq f(x, z, g, v).$$

Proof. Given a feasible solution (x, z, g, v) of (DMC) and another feasible assignment $\tilde{v}$ of the circulation decisions, we define

$$\mathcal{A}_{\text{fix}} := \{a \in \mathcal{A}_{\text{change}} : z_a = 0\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : g_{ij} = 0\} \cup \{(i,j) \in \mathcal{A}_{\text{head}} : \tilde{v}_{ij} = 0\}$$

and compute a disposition timetable $\tilde{x}$ by applying algorithm Cpm. As Cpm computes a time-minimal timetable and since (5.21) is satisfied, $\tilde{x}_j \leq x_j$ for all $j \in \mathcal{E}$. Hence, $f(\tilde{x}, z, g, \tilde{v}) \leq f(x, z, g, v)$. □

The statement of this lemma is the following: If we can change the circulation decisions in such a way that we improve the "right" side of the matching problem, i.e. if we can reduce the time of some events in $\mathcal{E}_{\text{start}}$ without increasing it for others, then this local change improves the whole solution.

In general, algorithm Local-Improvement does not compute an optimal solution of (DMC) – however, by fixing the circulation decisions in Step 1 as they are in the original plan, Local-Improvement never computes a solution that is worse than in the case of (DM) with fixed circulations where no changes to the rolling stock circulations are allowed at all (if the same delay management strategy is used), but it might improve the solution significantly in some cases.

Another approach for solving the problem is to use the known branch and bound approaches for the delay management problem (see for example [Sch06] and [Job08]) where lower bounds can be derived by solving relaxations of (DMC), while algorithm Local-Improvement yields upper bounds. Within the branch and bound, we can additionally use the circulation variables for branching.

Chapter 6

Robustness Aspects

In the previous chapters, we investigated different strategies for delay management and focused on the operational phase (when the trains are on the track) and on how to *react* when delays occur. However, it makes sense to already take into account delays during the strategic *planning* phase to reduce the probability and the propagation of delays. One possibility is to already take into account robustness when planning the lines. To make a line plan robust against delays, one approach is to distribute the traffic evenly across the network, see for example [SS06]. However, research on line planning goes beyond the scope of this work.

Other approaches consider robustness aspects when computing the timetable and try to add slack times to the timetable in a "smart" way to make it robust against small delays. In Section 6.1, we shortly resume results from a joint research project on computing delay resistant railway timetables; it has been published in [LSS+10]. In Sections 6.2-6.5, we present the concept of recoverable robustness, its extension to multi-stage recoverable robustness and the application of both concepts to robust timetabling. We conclude our investigation of recoverable robustness by summarizing the results in Section 6.6. Sections 6.2-6.6 are based on a joint research project and partly have been published in [CDSS08] and [CDD+09b].

6.1 Computing Delay Resistant Railway Timetables

As mentioned in the overview of related work, many different approaches on robust timetabling exist in the literature. In short, we subsume some results from a joint research project, a case study on robust timetabling, published in [LSS+10]. There, the focus is on taking into account delay management already in the objective of the

timetabling step: Instead of minimizing the weighted sum of the durations of the activities, we added a second term to the objective to count the *expected* delay. Of course the expected delay depends not only on the distribution of the source delays, but also on the delay management strategy that is applied during the operational phase. As periodic timetabling and delay management are both already NP-hard, it is too ambitious to take into account an *optimal* delay management strategy in the objective of timetabling. Hence, we assumed a simplified delay management strategy, namely a strict no-wait policy. Then, for different distributions of the source delays, we computed robust timetables, according to the modified objective function, evaluated their robustness under optimal delay management and compared them to an optimal, non-robust timetable. It turned out that the simplified delay management in the objective of timetabling is a good estimate of the optimal delay management strategy. Furthermore, the results imply that we can significantly reduce the passengers' delays at a rather small price of robustness, i.e. the increase of nominal travel times is rather small, compared to the level of robustness which we achieved. For further details on this approach, we refer to [LSS+10].

We now introduce the concepts of recoverable robustness and multi-stage recoverable robustness and show how to apply them to the timetabling problem.

6.2 Recoverable Robustness

First, we present the basic concept of recoverable robustness which we use for robust timetabling and which we extend to the multi-stage case in the next sections.

As already mentioned in the introduction of this work, optimization in public transportation – as well as optimization for many other real-world applications – consists of two parts: A *strategic planning phase* which aims for a good utilization of available resources and an *operational phase* where a good reaction to unforeseen disturbances is required once the system is running. The strategic planning phase usually starts long before the system starts to operate, while in the operational phase, an immediate response to unexpected events is required.

In the past, many different approaches for robust optimization have been presented. Most of them are based on *stochastic programming* or on *robust optimization*. Stochastic programming aims for computing solutions that either minimize the expected costs (multi-stage stochastic programming, recourse models) or satisfy the constraints with a high probability (chance constrained programming). However, stochastic programming requires extensive knowledge of the probability distribution of disturbances, and the resulting stochastic programs might be very hard to solve.

In contrast to stochastic programming, robust optimization is purely deterministic. In what we call *strict robustness*, the solution not only has to be feasible for the original input instance, but also for each scenario from the (possibly limited) set of allowed scenarios (modeling the expected disturbances of the original input), and it might not be changed when the actual scenario gets known. This is a major drawback: As the initial solution of the optimization problem has to feasible for a large set of possible modifications of the original input, it might be far too conservative, and it does not take into account recovery facilities that might be available.

The idea of recoverable robustness now is to explicitly take into account available recovery facilities. The solution of an optimization problem no longer has to be feasible for a whole set of admissible disturbances, but we require that it can be *recovered* to get a feasible solution. This concept fits very well for public transportation: in practice, the probability distribution of disturbances is not known, so stochastic programming cannot be applied. Strict robustness would lead to solutions that are far too conservative and too expensive (as large slack times would have to be added to almost all activities, dramatically increasing the passengers' traveling times). Recoverable robustness however is some kind of "natural" approach to robust timetables: compute the timetable in such a way that the effect of delays in the operational phase is limited when applying a given repair strategy.

The original concept of recoverable robustness has been introduced in [LLMS09] and further refined in [CDD+07]. In this section, we resume the concept and the basic notation, based on our presentation in [CDSS08].

In the concept of recoverable robustness, disruptions are modeled as changes in the input data. For each disruption we consequently obtain a new (disturbed) instance of the original optimization problem. Without loss of generality, we consider minimization problems P characterized by the following parameters:

- By I, we denote the set of instances of the optimization problem P.
- $F(i)$ represents the set of all feasible solutions for a given instance $i \in I$.
- Finally, $f\colon S \to \mathbb{R}^{>0}$ is the objective function of the optimization problem P that has to be minimized; here, the set

$$S := \bigcup_{i \in I} F(i)$$

  is the set of all feasible solutions of P for all potential input instances.

In recoverable-robust optimization, we do not only want to find a good or even optimal solution for some given initial instance $i \in I$, but a *robust* one. Hence, additional concepts for describing the robustness are needed:

- The function $M : I \to 2^I$, a *modification* function for instances of P, models disruptions of the current scenario. If $i \in I$ is an instance (or scenario) of problem P, a *disruption* is a modification of i leading to another instance $i' \in I$. However, such a modification i' is usually not completely different from i. In order to model this fact, we use the modification function M to define the set $M(i)$ as the set of all instances which are modifications of the instance i, i.e. $M(i)$ contains all instances of P that might occur if instance i is disturbed. Figuratively speaking, it might be much easier to compute robust solutions if M is rather restrictive, i.e. if only small changes in the input are allowed.

  If $s \in F(i)$ dennotes the planned solution for the instance $i \in I$ and a disturbance $i' \in M(i)$ occurs, then a new solution $s' \in F(i')$ for P has to be computed.

- $\mathbb{A}_{\text{rec}}$ is a class of *recovery algorithms* for P. Each algorithm $A_{\text{rec}} : S \times I \to S$ from this class works as follows: given a solution $s^0 \in S$ of P for an instance $i^0 \in I$ and a modification $i^1 \in M(i^0)$ of this instance, A_{rec} computes $A_{\text{rec}}(s^0, i^1) = s^1$ where $s^1 \in F(i^1) \subseteq S$ represents the recovered solution of P. In other words, given the original solution s^0 and a disturbed instance i^1, A_{rec} computes a new solution which is feasible for the disturbed instance.

  In general, a class of recovery algorithms $\mathbb{A}_{\text{rec}}$ is defined in terms of some type of *limitation*. In the following we provide three examples for the class $\mathbb{A}_{\text{rec}}$.

  $\mathbb{A}^{\mathbf{0}}_{\mathbf{rec}}$: strict robustness. This class models the case in which no recovery is allowed or no recovery capabilities are available. Hence, the initial solution has to be feasible for all admissible disturbances and might not be changed if the actual instance gets known. Mathematically, each algorithm $A_{\text{rec}} \in \mathbb{A}^0_{\text{rec}}$ fulfills the following constraint:

  $$A_{\text{rec}}(s^0, i^1) = s^0 \quad \forall i^0 \in I,\ \forall s^0 \in S,\ \forall i^1 \in M(i^0). \tag{6.1}$$

  $\mathbb{A}^{\mathbf{1}}_{\mathbf{rec}}$: bounded distance from the original solution. The class $\mathbb{A}^1_{\text{rec}}$ is defined by imposing a constraint on the solutions provided by the recovery algorithms in $\mathbb{A}^1_{\text{rec}}$. In particular, the recovered solutions computed by these recovery algorithms must not deviate "too much" from the original solution s, according to a distance measure d. Formally, given a real number $\Delta \in \mathbb{R}^{\geq 0}$ and a distance function $d : S \times S \to \mathbb{R}^{\geq 0}$, each element A_{rec} in such a class fulfills the following constraint:

  $$d(s^0, A_{\text{rec}}(s^0, i^1)) \leq \Delta \quad \forall i^0 \in I,\ \forall s^0 \in S,\ \forall i^1 \in M(i^0). \tag{6.2}$$

Note that $\mathbb{A}^0_{\text{rec}} \subset \mathbb{A}^1_{\text{rec}}$: If, for any distance function d, $\Delta = 0$, then constraints (6.1) and (6.2) are equivalent.

$\mathbb{A}^2_{\text{rec}}$: bounded computational power. The class $\mathbb{A}^2_{\text{rec}}$ is defined by bounding the computational power of the recovery algorithms. Formally, given a function $t : I \times S \times I \to \mathbb{N}$, each element $A_{\text{rec}} \in \mathbb{A}^2_{\text{rec}}$ fulfills the following constraint:

$$\forall i^0 \in I,\ \forall s^0 \in S,\ \forall i^1 \in M(i^0),$$
$$A_{\text{rec}}(s^0, i^1) \text{ must be computed in time } \mathcal{O}(t(i^0, s^0, i^1)).$$

Of course combinations of these classes are also possible.

We first recall the basic definitions for recoverable-robust optimization before we apply the concept of recoverable robustness to timetabling problems in Section 6.3.

First, we define what we mean by a recoverable robustness problem and by a feasible solution of such a problem:

Definition 6.1. *A* recoverable robustness problem *is defined by the triple* $(P, M, \mathbb{A}_{\text{rec}})$. *All the recoverable robustness problems form the class* RRP.

Definition 6.2. *Let* $\mathcal{P} = (P, M, \mathbb{A}_{\text{rec}}) \in \text{RRP}$. *Given an instance* $i^0 \in I$ *of* P, *an element* $s^0 \in F(i)$ *is a* feasible solution *for* i^0 *with respect to* $\mathcal{P}$ *if and only if the following relationship holds:*

$$\exists A_{\text{rec}} \in \mathbb{A}_{\text{rec}} : \forall i^1 \in M(i^0),\ A_{\text{rec}}(s^0, i^1) \in F(i^1).$$

If s^0 *is feasible with respect to the recoverable robustness problem* $\mathcal{P}$, *we call it* robust solution *with respect to the original optimization problem* P.

In other words, $s^0 \in F(i^0)$ is feasible for i^0 with respect to $\mathcal{P}$ if, for each possible disruption $i^1 \in M(i^0)$, it can be *recovered* by applying some (fixed) algorithm $A_{\text{rec}} \in \mathbb{A}_{\text{rec}}$. We use the notation $F_{\mathcal{P}}(i^0)$ to denote all feasible solutions for i^0 with respect to the recoverable robustness problem $\mathcal{P}$ (i.e. all robust solutions with respect to the original problem P). Formally,

$$F_{\mathcal{P}}(i^0) := \{s^0 \in F(i^0) : s^0 \text{ is a feasible solution for } i^0 \text{ with respect to } \mathcal{P}\}.$$

A possible scenario for this situation is depicted in Figure 6.1. As can be seen there, the set $F_{\mathcal{P}}(i)$ can also be considered as the set of those solutions for the instance i that can be "recovered" if a disturbance occurs. Finding robust solutions is the task of *robust algorithms* which we define in the following definition.

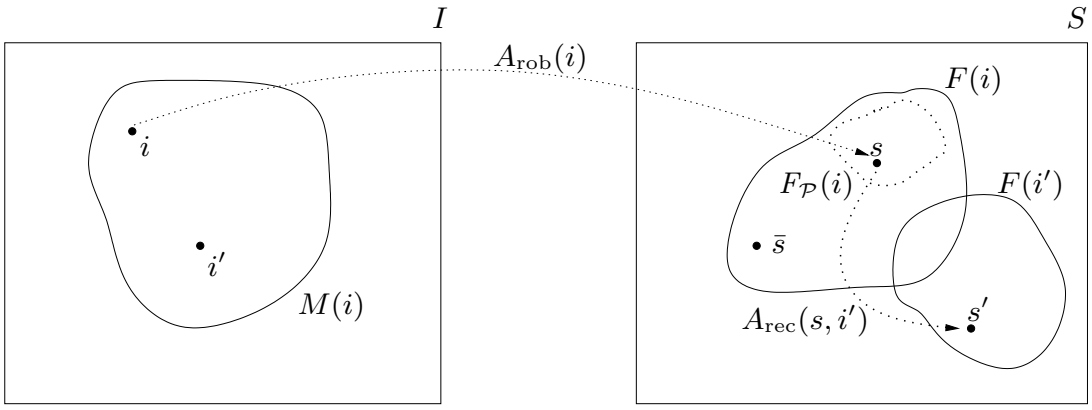

Figure 6.1: An example for the single-stage model: I and S are the set of instances and the set of solutions of P. $M(i)$ is the set of instances obtainable after a modification of the initial instance i, while $F(i)$ and $F(i')$ denote the set of feasible solutions for i and for i' with respect to P. $\bar{s} \in F(i)$ is the *optimal*, but non-robust solution for i with respect to P; $F_{\mathcal{P}}(i)$ is the set of all feasible solutions for i with respect to $\mathcal{P}$, i.e. the set of all robust solutions for i with respect to P. s is a robust solution, obtained by A_{rob}, and s' is the recovered solution obtained by $A_{\text{rec}} \in \mathbb{A}_{\text{rec}}$ after the disruption $i' \in M(i)$.

Definition 6.3. *Given a recoverable robustness problem* $\mathcal{P} = (P, M, \mathbb{A}_{\text{rec}}) \in \text{RRP}$*, a* robust algorithm *for* $\mathcal{P}$ *is any algorithm* $A_{\text{rob}} : I \to S$ *such that, for each* $i \in I$*,* $A_{\text{rob}}(i)$ *is robust for* i *with respect to* P*.*

Note that A_{rob} as in the definition above provides the solution s^0 as defined in Definition 6.2. In the case of strict robustness, a robust algorithm A_{rob} for $\mathcal{P}$ must provide a solution s^0 for i^0 such that for each possible modification $i^1 \in M(i^0)$, we have $s^0 \in F_{\mathcal{P}}(i^1)$. The meaning is the following: If A_{rec} has no recovery capability, then A_{rob} has to find solutions that "absorb" *any* possible disturbance.

For each instance $i \in I$ of the initial problem P, the price of robustness of the robust algorithm A_{rob} is given by the maximum ratio between the cost of the solution provided by A_{rob} and the cost of an optimal (non-robust) solution.

Definition 6.4. *The* price of robustness *of a robust algorithm* A_{rob} *for a recoverable robustness problem* $\mathcal{P} \in \text{RRP}$ *is given by*

$$P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) := \max_{i \in I} \left\{ \frac{f(A_{\text{rob}}(i))}{\min\{f(x) : x \in F(i)\}} \right\}.$$

The price of robustness of a recoverable robustness problem $\mathcal{P} \in \text{RRP}$ is then given by the minimum price of robustness among all possible robust algorithms for this problem. Formally:

Definition 6.5. *The* price of robustness *of a recoverable robustness problem* $\mathcal{P} \in \text{RRP}$ *is given by*

$$P_{\text{rob}}(\mathcal{P}) = \min\{P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) : A_{\text{rob}} \textit{ is a robust algorithm for } \mathcal{P}\}.$$

If there are different robust algorithms for a recoverable robustness problem, we want to identify the "best" one:

Definition 6.6. *Let* $\mathcal{P} \in \text{RRP}$ *and let* A_{rob} *be a robust algorithm for* $\mathcal{P}$*. Then,*

- A_{rob} *is called* $\mathcal{P}$-optimal *if* $P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) = P_{\text{rob}}(\mathcal{P})$*;*
- A_{rob} *is called* exact *if* $P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) = 1$.

We call a solution computed by an optimal algorithm $\mathcal{P}$-optimal*, while a solution computed by an exact algorithm is called* exact*.*

Note that each exact algorithm is optimal.

6.3 Recoverable-Robust Timetabling

Having introduced the concept of recoverable robustness, we now apply it to a (simplified variant) of the timetabling problem to compute robust timetables which can be recovered if a delay occurs. To this end, we first introduce the specific timetabling problem which we consider and describe the delay scenarios and the restrictions for the recovery we are looking at. Afterwards, we provide a general solution approach for computing robust timetables.

In the following, we focus on directed acyclic graphs instead of event-activity networks as an event-activity network is a special case of a DAG, while some of the results hold for the more general structure of a directed acyclic graph. So let $G = (V, A)$ be a DAG where – referring to the notation from the previous chapters – we call the nodes in V events and the edges in A activities. We consider a simplified timetabling problem where we neglect wait/depart decisions and headways. In addition, we consider only *non-periodic* timetabling.

The simplified timetabling problem we are interested in is the following: We are looking for a timetable $\pi : V \to \mathbb{R}^{\geq 0}$ assigning a point of time to each event $u \in V$, respecting

the minimal duration of each activity. However, in general we also allow negative weights $w : V \to \mathbb{R}$. The initial timetabling problem TT can be stated as

$$(\boldsymbol{TT}) \quad \min f(\pi) = \sum_{u \in V} w_u \pi_u$$

such that

$$\pi_v - \pi_u \geq L_a^0 \qquad \forall a = (u, v) \in A \tag{6.3}$$

$$\pi_u \in \mathbb{R}^{\geq 0} \qquad \forall u \in V. \tag{6.4}$$

An instance i of TT is specified by $i = (G, w, L^0)$. In contrast to periodic timetabling which is NP-hard, TT can be solved in polynomial time by linear programming. We consider two special cases of TT, both having the same constraints (6.3) and (6.4), but differing in the objective functions.

timetabling with arc weights ($\boldsymbol{TT^a}$): Here we consider the objective which is usually used in timetabling, namely to minimize the weighted sum of the duration of all activities (or equivalently to minimize the weighted sum of the slack times of all activities). The objective of this problem is

$$(\boldsymbol{TT^a}) \quad \min f_{\text{arcs}}(\pi) = \sum_{a=(u,v) \in A} w_a(\pi_v - \pi_u)$$

with $w_a \in \mathbb{R}^{\geq 0}$ for all $a \in A$. It is a special case of TT, namely if

$$w_u = \sum_{a=(v,u) \in A} w_a - \sum_{a=(u,v) \in A} w_a$$

for each $u \in V$.

timetabling with nonnegative node weights ($\boldsymbol{TT^v}$): Here we consider the problem TT but require $w_u \geq 0$ for all $u \in V$ as we did in the previous chapters.

To compute an optimal solution of both problems, in most cases, algorithm Cpm can be used as stated in the following two lemmas:

Lemma 6.7. *Given an instance $i = (G, w, L^0)$ of TT^v,* Cpm *computes an optimal solution.*

Lemma 6.7 is a direct consequence of the definition of algorithm Cpm – by construction, it schedules each event as early as possible, hence minimizing the weighted times as the weights w_u in problem TT are nonnegative.

Lemma 6.8. *Given an instance $i = (G, w, L^0)$ of TT^a,* Cpm *computes an optimal solution if G is a tree.*

Note that this does not need to be true if G is not a tree: it often makes sense to schedule events that do *not* belong to the critical path (i.e. a path without slack times) later than necessary to avoid slack on activities with high weights. An example is depicted in Figure 6.2: In that example, the solution obtained from Cpm is $\pi_1 = 0$, $\pi_2 = \pi_3 = 1$ and $\pi_4 = 3$ with an objective value of 8. However, in an optimal solution, $\pi_3 = 2$ with an objective value of only 7 as the weight of activity $(3, 4)$ is larger than the weight of activity $(1, 3)$.

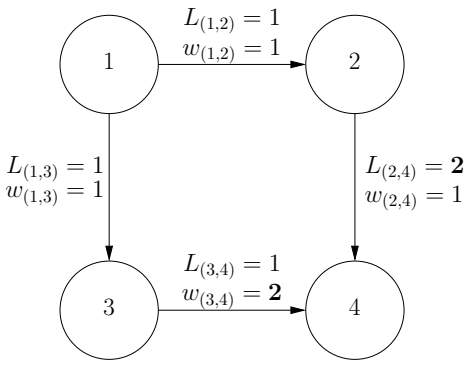

Figure 6.2: A solution for problem TT^a, computed by the Critical Path Method, might not be optimal if the graph is not a tree.

However, if G is a tree, scheduling each event as early as possible also minimizes all slack times.

In the following, we show how to apply the concept of recoverable robustness to the timetabling problem TT. To turn the timetabling problem TT into a recoverable robustness problem, we have to define a modification function M and a class of recovery algorithm $\mathbb{A}_{\text{rec}}$.

- The modification function M for an instance $i^0 = (G, w, L^0)$ and a constant $\alpha \in \mathbb{R}^{>0}$ is defined as

$$M(i^0) = \left\{(G, w, L^1) : \exists \overline{a} \in A : L^0_{\overline{a}} < L^1_{\overline{a}} \leq L^0_{\overline{a}} + \alpha \text{ and } L^1_a = L^0_a \ \forall a \neq \overline{a}\right\}.$$

Hence, we represent the delay of an activity a by increasing the initial value L_a^0 to some value $L_a^1 > L_a^0$. The definition ensures that only one single delay on some activity $\bar{a}$ is allowed and bounded by α while the lower bounds of all other activities are fixed.

- For the set of allowed recovery algorithms $\mathbb{A}_{\text{rec}}$, we investigate the following two limitations:

  limited-events ($\mathbb{A}_{\Delta}$): Here, we assume that there are resources to change the time of a limited number of events only. In particular, if π is a solution for TT and x^1 is a disposition timetable computed by any recovery algorithm in $\mathbb{A}_{\text{rec}}$, then x^1 must satisfy

  $$d(x^1, \pi) := \left|\{u \in V : x_u^1 \neq \pi_u\}\right| \leq \Delta$$

  for some given $\Delta \in \mathbb{N}$. This class of recovery algorithms is denoted by $\mathbb{A}_{\Delta}$.

  limited-delay ($\mathbb{A}_{\delta}$): As second limitation of the recovery algorithms, we again require that x^1 must not deviate "too much" from the initial timetable π, but this time we consider the sum of all deviations of all events, i.e. x^1 must satisfy

  $$d(x^1, \pi) := \left\|x^1 - \pi\right\|_1 \leq \delta$$

  for some given $\delta \in \mathbb{N}$. This class of recovery algorithms is denoted by $\mathbb{A}_{\delta}$.

  Both classes $\mathbb{A}_{\Delta}$ and $\mathbb{A}_{\delta}$ belong to the general class $\mathbb{A}_{\text{rec}}^1$. Note that we can restrict both classes to strict robustness by setting $\Delta := 0$ and $\delta := 0$.

Using those definitions, the initial problem TT can be turned into a recoverable robustness problem $\mathcal{TT}_1 = (TT, M, \mathbb{A}_{\text{rec}}) \in \text{RRP}$. Special cases of $\mathcal{TT}_1$ can be obtained by replacing TT with TT^a or TT^v and choosing a class of recovery algorithms. Among the four possible combinations, in the remainder of this chapter, we focus on the following two recoverable robustness problems:

Robust timetabling with nonnegative node weights and limited-events. In the definition of this problem, we use the special timetabling problem TT^v with nonnegative node weights and the class $\mathbb{A}_{\Delta}$ of recovery algorithms. The resulting recoverable robustness problem hence is

$$\mathcal{TT}_1^v := (TT^v, M, \mathbb{A}_{\Delta}).$$

Robust timetabling with arc weights and limited-delay. Here, we consider the special timetabling problem TT^a with objective function f_{arcs} together with the class $\mathbb{A}_\delta$ of recovery algorithms. The resulting recoverable robustness problem is

$$\mathcal{TT}_1^a := (TT^a, M, \mathbb{A}_\delta).$$

Solving these problems requires to define robust algorithms as stated in Definition 6.3. For example, if we consider problem $\mathcal{TT}_1^v$, then we need to construct a robust algorithm A_{rob} such that, for a given instance $i^0 = (G, w, L^0)$ of TT^v, A_{rob} computes a timetable π with

$$\pi_v - \pi_u \geq L_a^0 \quad \forall a = (u, v) \in A$$

(this ensures *feasibility*) and such that for all modifications $(G, w, L^1) \in M(i^0)$ of i^0, there exists a disposition timetable x^1 with

$$x_v^1 - x_u^1 \geq L_a^1 \qquad \forall a = (u, v) \in A \tag{6.5}$$

$$x_u^1 \geq \pi_u \qquad \forall u \in V \tag{6.6}$$

$$d(x^1, \pi) \leq \Delta \tag{6.7}$$

(this ensures *robustness*). Note that if we consider problem $\mathcal{TT}_1^a$ instead of $\mathcal{TT}_1^v$, everything we have to change is constraint (6.7) where we have to replace Δ by δ.

To compute robust solutions, we use the following general solution approach: first add an additional slack time $s_a \geq 0$ to the lower bound L_a^0 of each activity $a \in A$, then compute an optimal solution of the resulting instance and take it as a robust solution. Formally, we obtain an algorithm Alg_s^+ for each vector $s \in \mathbb{R}^{|A|}$:

Algorithm 6.1: Alg_s^+

Input: An instance $i^0 = (G, w, L^0)$ of TT and a vector s with $s_a \geq 0 \; \forall \; a \in A$.

Step 1: Define $\bar{L}_a := L_a^0 + s_a$ for all $a \in A$.

Step 2: Solve $\bar{i} = (G, w, \bar{L})$ optimally.

Instead of adding a positive slack time to the lower bounds L_a, one can also multiply them with factors $t_a \geq 1$. This approach is known as *proportional buffering* (see [LS09]) and leads to the following algorithm Alg_t^* which differs from Alg_s^+ only in Step 1.

Algorithm 6.2: Alg_t^*

Input: An instance $i^0 = (G, w, L^0)$ of TT and a vector t with $t_a \geq 1 \; \forall \; a \in A$.

Step 1: Define $\bar{L}_a := t_a \cdot L_a^0$ for all $a \in A$.

Step 2: Solve $\bar{i} = (G, w, \bar{L})$ optimally.

According to Lemma 6.7, in the case of TT^v, Step 2 of both algorithms can be done in linear time by the Critical Path Method for an arbitrary DAG. If we consider problem TT^a, according to Lemma 6.8, Step 2 of both algorithms can be done efficiently by the Critical Path Method when G is a tree. Otherwise, linear programming can be used, yielding a polynomial runtime.

For the recoverable robustness problems which we consider, algorithms Alg_s^+ and Alg_t^* are robust if s is large enough. Moreover, the price of robustness increases in s. We collect some properties of both algorithms in the following lemma.

Lemma 6.9. *Consider Alg_s^+ and Alg_t^* as algorithms for solving $\mathcal{TT}_1^v$. Then, the following holds:*

1. *Alg_s^+ is robust for $\Delta = 0$ (strict robustness) if and only if $s_a \geq \alpha$ for each $a \in A$.*
   Alg_t^* is robust for $\Delta = 0$ (strict robustness) if $t_a \geq \frac{L_a^0 + \alpha}{L_a^0}$ for each $a \in A$.
2. *Let Alg_s^+ be robust. Then $\mathsf{Alg}_{s'}^+$ is robust if $s'_a \geq s_a$ for all $a \in A$.*
   Let Alg_t^* be robust. Then $\mathsf{Alg}_{t'}^*$ is robust if $t'_a \geq t_a$ for all $a \in A$.
3. *The price of robustness of Alg_s^+ is monotone in s. In particular:*
   Let $\mathsf{Alg}_{s^1}^+$ and $\mathsf{Alg}_{s^2}^+$ be robust and let $s_a^2 \geq s_a^1 \geq 0$ for all $a \in A$. Then $P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_{s^1}^+) \leq P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_{s^2}^+)$.
   The price of robustness of Alg_t^* is monotone in t. In particular:
   Let $\mathsf{Alg}_{t^1}^*$ and $\mathsf{Alg}_{t^2}^*$ be robust and let $t_a^2 \geq t_a^1 \geq 1$ for all $a \in A$. Then $P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_{t^1}^*) \leq P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_{t^2}^*)$.

If we replace $\mathcal{TT}_1^v$ by $\mathcal{TT}_1^a$ and Δ by δ, 1-3 still hold.

Proof. For the first statement, note that according to our definition of the modification function M, no activity can have a delay greater than α. A slack of $s_a \geq \alpha$ hence reduces each possible delay completely such that $x^1 = \pi$ is a feasible disposition timetable. The same holds if $t_a \geq \frac{L_a^0 + \alpha}{L_a^0}$ since $t_a \cdot L_a^0 \geq L_a^0 + \alpha$ for all $a \in A$.

To show that Alg_s^+ is strictly robust only if $s_a \geq \alpha$ for each $a \in A$, let us assume that π is a robust solution computed by Alg_s^+ with $\pi_v - \pi_u - L_a^0 < \alpha$ for some $a = (u, v) \in A$. Now, if a modification $i^1 \in M(i^0)$ of i^0 satisfies $L_a^1 = L_a^0 + \alpha$, i.e. a delay of α occurs on a, then $x_v^1 - x_u^1 = \pi_v - \pi_u < L_a^0 + \alpha = L_a^1$. Hence, constraint (6.5) is violated – a contradiction to the assumption that π is robust.

The second statement is clear as adding additional slack cannot turn a robust solution into a nonrobust one – it only increases the objective value.

To obtain the monotonicity claimed in the third statement, we have to show that $f(\mathsf{Alg}_{s^1}^+(i)) \leq f(\mathsf{Alg}_{s^2}^+(i))$ (and $f(\mathsf{Alg}_{t^1}^*(i)) \leq f(\mathsf{Alg}_{t^2}^*(i))$, respectively) which is true due to the monotonicity of the tensions of the timetables corresponding to the slack times s_a (or $(t_a - 1)L_a^0$). □

Although computing a robust solution of the initial problem does not require to explicitly specify a recovery algorithm, for $\mathcal{TT}_\sigma^a$ and $\mathcal{TT}_\sigma^v$, we can provide such a recovery algorithm as in the following. Assume that π is a solution computed by any robust algorithm A_{rob} with respect to the instance i^0. Let $i^1 = (G, w, L^1) \in M(i^0)$. As π is a robust solution, we know that a disposition timetable x^1 satisfying (6.5)-(6.7) exists. It can be computed by using the Critical Path Method as recovery algorithm. CPM computes that disposition timetable x^1 with the minimum value of $d(x^1, \pi)$, i.e. it minimizes

$$\left|\{u \in V : x_u^1 \neq \pi_u\}\right| \text{ and } \left\|x^1 - \pi\right\|_1$$

at the same time. Hence it is able to recover (if a recovery solution exists) or to find out that such a solution does not exist. Additionally, among all timetables satisfying constraints (6.5)-(6.7), CPM provides the disposition timetable with the optimal value for TT^v and, if G is a tree, also for TT^a.

Unfortunately, the following complexity results hold:

Theorem 6.10 ([CDSS08, CDD+09b, DDNP09]). *The problem of computing $P_{\text{rob}}(\mathcal{TT}_1^v)$ is NP-hard. If the underlying DAG is a tree, the problem remains NP-hard. If Δ is fixed and not part of the input, the problem remains NP-hard for any $\Delta \geq 3$.*

Theorem 6.11 ([CDD+08, CDD+09a]). *The problem of computing $P_{\text{rob}}(\mathcal{TT}_1^a)$ is NP-hard. Furthermore, if Δ is fixed and not part of the input, the problem remains NP-hard for any $\Delta \geq 3$ on an arbitrary DAG and for any $\Delta \geq 5$ if the underlying DAG is an event-activity network.*

As a consequence, it is unlikely to find a polynomial-time algorithm for computing an optimal solution of $\mathcal{TT}_1$:

Corollary 6.12. *Unless $P = NP$, there exists no polynomial-time algorithm for solving $\mathcal{TT}_1$, $\mathcal{TT}_1^v$ or $\mathcal{TT}_1^a$ to optimality.*

Hence, in the following, we focus on general results and compute optimal robust solutions only for special cases of problems $\mathcal{TT}_1^v$ and $\mathcal{TT}_1^a$.

6.3.1 Limited-Events and Timetabling with Node Weights

In this section, we consider the recoverable robustness problem $\mathcal{TT}_1^v = (TT^v, M, \mathbb{A}_\Delta)$. We start with the observation that each timetime is robust if Δ is "large enough", compared to the size of the event-activity network:

Lemma 6.13. *If $\Delta \geq |V| - 1$, then each feasible timetable is robust.*

Hence, in the following, we always assume $\Delta \leq |V| - 2$.

First, we derive some results for the case of strict robustness, i.e. for the case $\Delta = 0$. Afterwards, we drop this restriction, but assume that the underlying DAG is a linear graph.

Strict robustness.

For the case of strict robustness, we prove some results on the price of robustness of special versions of algorithms Alg_s^+ and Alg_t^* (Algorithm 6.1 and Algorithm 6.2 from Section 6.3) which either assign the same constant amount of slack to each arc or multiply the lower bound of each arc with the same constant factor. To this end, let $L_{\min}$ be the minimum value of function L with respect to all admissible instances of $\mathcal{TT}_1^v$. Denoting by α the maximum delay from the modification function M of $\mathcal{TT}_1^v$ and defining

$$\gamma := 1 + \frac{\alpha}{L_{\min}},$$

we set

$$s = (\alpha, \alpha, \ldots, \alpha) \in \mathbb{R}^{|A|}$$
$$t = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{R}^{|A|}.$$

Then, we use algorithms Alg_s^+ and Alg_t^* with s and t as defined above to compute robust solutions; we denote the resulting algorithms by Alg_α^+ and Alg_γ^* to emphasize that they assign the same constant amount of slack to each arc or multiply the lower bound of each arc with the same constant factor, respectively. Then, the following holds:

Corollary 6.14. Alg_α^+ *and* Alg_γ^* *are robust algorithms for* $\mathcal{TT}_1^v$.

Proof. For Alg_α^+, this is a direct consequence of Lemma 6.9. As

$$\gamma = 1 + \frac{\alpha}{L_{\min}} \geq 1 + \frac{\alpha}{L_a^0} = \frac{L_a^0 + \alpha}{L_a^0} \quad \forall a \in A,$$

due to Lemma 6.9, the claim also holds for Alg_γ^*. □

The following lemma shows the price of robustness of Alg_γ^*.

Lemma 6.15. *Let* $\mathcal{TT}_1^v$ *be defined with* $\Delta = 0$. *Then*

$$P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_\gamma^*) = 1 + \frac{\alpha}{L_{\min}}.$$

Proof. Let $i = (G, L, w)$ be an instance of $\mathcal{TT}_1^v$. Denoting by π^γ the solution provided by $\mathsf{Alg}_\gamma^*(i)$ and by π the optimal (non-robust) solution provided by Cpm, we show that for each $v \in V$, $\pi_v^\gamma = \gamma\pi_v$. By contradiction, assume $\pi_v^\gamma \neq \gamma\pi_v$ for some $v \in V$ such that π_v^γ is minimal among all such events. Clearly, v must be different from v_1, hence there exists an activity $a = (u, v) \in A$ such that $\pi_v^\gamma = \pi_u^\gamma + \gamma L_a$. As π_v^γ is minimal and $\pi_u^\gamma < \pi_v^\gamma$, we have $\pi_u^\gamma = \gamma\pi_u$. Thus $\pi_v^\gamma = \gamma\pi_u + \gamma L_a = \gamma\pi_v$, a contradiction. Hence $\pi_v^\gamma = \gamma\pi_v$ for each $v \in V$. Then, using the definition of the price of robustness,

$$\begin{aligned}
P_{\text{rob}}(\mathcal{TT}_1^v, \mathsf{Alg}_\gamma^*) &= \max_{i \in I} \left\{ \frac{f(\mathsf{Alg}_\gamma^*(i))}{\min\{f(x) : x \in F(i)\}} \right\} \\
&= \max_{i=(G,L,w)\in I} \frac{\sum_{u\in V} w_u \pi_u^\gamma}{\sum_{u\in V} w_u \pi_u} \\
&= \max_{i=(G,L,w)\in I} \frac{\sum_{u\in V} w_u \gamma\pi_u}{\sum_{u\in V} w_u \pi_u} \\
&= \gamma \\
&= 1 + \frac{\alpha}{L_{\min}}.
\end{aligned}$$

□

Lemma 6.16. *For each instance* $i \in I$, $f(\mathsf{Alg}_\alpha^+(i)) \leq f(\mathsf{Alg}_\gamma^*(i))$.

Proof. Let $i = (G, L, w) \in I$, and let π^γ and π^α denote the solutions computed by $\mathsf{Alg}_\gamma^*(i)$ and $\mathsf{Alg}_\alpha^+(i)$. To prove the statement, it is sufficient to show that

$$\pi_u^\alpha \leq \pi_u^\gamma \quad \forall\ u \in V$$

as the weights of all nodes are nonnegative. However, this is a direct consequence of

$$\gamma L_a = \left(1 + \frac{\alpha}{L_{\min}}\right) L_a = L_a + \alpha \frac{L_a}{L_{\min}} \geq L_a + a \quad \forall a \in A,$$

i.e. the slack time which algorithm Alg_γ^* adds to each activity is always larger than or equal to the slack time which algorithm Alg_α^+ adds. □

Lemmas 6.15 and 6.16 imply the following result concerning the price of robustness of algorithm Alg^+_α:

Corollary 6.17. *Let $\mathcal{TT}^v_1$ be defined with $\Delta = 0$. Then,*

$$P_{\text{rob}}(\mathcal{TT}^v_1, \mathsf{Alg}^+_\alpha) \leq 1 + \frac{\alpha}{L_{\min}}.$$

Linear graphs.

Now, we drop the restriction to strict robustness and allow arbitrary Δ; however, we now assume that the DAG is a linear graph. This assumptions allows us to present an optimal algorithm for $\mathcal{TT}^v_1$. The idea of the algorithm is to add each slack "as late as possible". Formally, we apply algorithm $\mathsf{Alg}^+_{s^\alpha}$ with a vector s^α defined by

$$s^\alpha_{a_j} := \begin{cases} \alpha & \text{if } (\Delta+1)|j \\ 0 & \text{else} \end{cases} \tag{6.8}$$

for all arcs $a_j \in A$, i.e. we add s^α_a to L^0_a for each $a \in A$ and calculate a solution of TT^v by applying Cpm, see Figure 6.3 for an illustration.

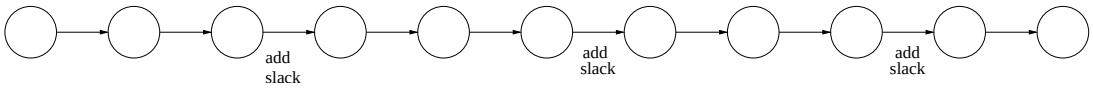

Figure 6.3: Adding additional slack time for the case $\Delta = 2$.

To show the robustness of $\mathsf{Alg}^+_{s^\alpha}$, we observe the following more general lemma, stating that a timetable π is robust if and only if the slack time of each $\Delta + 1$ consecutive arcs is large enough to let vanish the delay.

Lemma 6.18. *If $\Delta \leq |V| - 2$, a timetable π for a linear graph G is robust if and only if the slack times satisfy*

$$\sum_{k=0}^{\Delta} s_{a_{j+k}} \geq \alpha \quad \forall j = 1, \ldots, |A| - \Delta. \tag{6.9}$$

Proof. Equation (6.9) ensures that the delay vanishes after at most Δ events, no matter where a source delay occurs. On the other hand, if (6.9) does not hold, e.g. $\sum_{k=0}^{\Delta} s_{a_{j+k}} < \alpha$ for some $j \in \{1, \ldots, |A| - \Delta\}$, a source delay of α at a_j leads to more than Δ delayed events and shows that π is not robust. □

In the next theorem we show that $\mathsf{Alg}^+_{s^\alpha}$ is an optimal robust algorithm, i.e. it leads to the smallest possible price of robustness for $\mathcal{TT}^v_1$.

Theorem 6.19. *$\mathsf{Alg}^+_{s^\alpha}$ is an optimal robust algorithm for $\mathcal{TT}^v_1$. Furthermore, if $w_u > 0$ for all $u \in V$, the optimal robust timetable computed by $\mathsf{Alg}^+_{s^\alpha}$ is unique.*

Proof. The robustness of $\mathsf{Alg}^+_{s^\alpha}$ follows from Lemma 6.18. It remains to show that it is optimal. To this end, let π^* be the output of $\mathsf{Alg}^+_{s^\alpha}$ with slack s^α and π' another robust timetable with slack $s' \neq s^\alpha$. To compare the objective value $f(\pi^*)$ of π^* with the objective value $f(\pi')$ of π', we split the objective f into sums f_k over $\Delta + 1$ consecutive nodes and show that π^* is not worse than π' for each of those sums f_k. For each timetable π, we have

$$\begin{aligned} f(\pi) &= \sum_{u \in V} w_u \pi_u \\ &= \sum_{j=1}^{|V|} w_{v_j} \pi_{v_j} \\ &= w_{v_1}\pi_{v_1} + \sum_{j=2}^{(\Delta+1)+1} w_{v_j}\pi_{v_j} + \sum_{j=(\Delta+1)+2}^{2(\Delta+1)+1} w_{v_j}\pi_{v_j} + \sum_{j=2(\Delta+1)+2}^{3(\Delta+1)+1} w_{v_j}\pi_{v_j} + ... \\ &= w_{v_1}\pi_{v_1} + f_0(\pi) + f_1(\pi) + f_2(\pi) + ... \end{aligned}$$

with

$$f_k(\pi) := \sum_{j=k(\Delta+1)+2}^{(k+1)(\Delta+1)+1} w_{v_j}\pi_{v_j}, \quad k = 0, 1, 2, ...$$

Now let k be fixed. For the sake of simplicity, we set $l = l(k) = k(\Delta + 1) + 2$ to the lower and $u = u(k) = (k + 1)(\Delta + 1) + 1$ to the upper bound of the summation index j in f_k. To compare f_k for different timetables, we use $\pi_{v_{j+1}} = \pi_{v_j} + L_{a_j} + s_{a_j}$ iteratively and write f_k as

$$
\begin{aligned}
f_k(\pi) &= \sum_{j=l}^{u} w_{v_j}\pi_{v_j} \\
&= \quad w_{v_l}\left[\pi_{v_{l-1}} + (L_{a_{l-1}} + s_{a_{l-1}})\right] \\
&\quad + w_{v_{l+1}}\left[\pi_{v_{l-1}} + (L_{a_{l-1}} + s_{a_{l-1}}) + (L_{a_l} + s_{a_l})\right] \\
&\quad + w_{v_{l+2}}\left[\pi_{v_{l-1}} + (L_{a_{l-1}} + s_{a_{l-1}}) + (L_{a_l} + s_{a_l}) + (L_{a_{l+1}} + s_{a_{l+1}})\right] \\
&\quad + ... \\
&\quad + w_{v_u}\left[\pi_{v_{l-1}} + (L_{a_{l-1}} + s_{a_{l-1}}) + (L_{a_l} + s_{a_l}) + ... + (L_{a_{u-1}} + s_{a_{u-1}})\right] \\
&= \pi_{v_{l-1}} \sum_{j=l}^{u} w_{v_j} + \sum_{j=l-1}^{u-1} \sum_{p=j+1}^{u} w_{v_p}(L_{a_j} + s_{a_j}) \\
&= \quad \pi_{v_{l-1}} \sum_{j=l}^{u} w_{v_j} + \underbrace{\sum_{j=l-1}^{u-2} \sum_{p=j+1}^{u-1} w_{v_p} L_{a_j}}_{(a)} + \underbrace{\sum_{j=l-1}^{u-2} \sum_{p=j+1}^{u-1} w_{v_p} s_{a_j}}_{(b)} \\
&\quad + \underbrace{w_{v_u} \sum_{j=l-1}^{u-1} L_{a_j}}_{(c)} + w_{v_u} \sum_{j=l-1}^{u-1} s_{a_j}.
\end{aligned}
$$

Terms (a) and (c) are the same for all timetables on the same graph, so they cancel out each other if we compute the difference $f_k(\pi') - f_k(\pi^*)$. As π^* satisfies (6.8), term (b) vanishes for π^* (because $s^{\alpha}_{a_j} = \alpha$ for $j = l-2$ and for $j = u-1$, but $s^{\alpha}_{a_j} = 0$ for all $j = l-1, \ldots, u-2$). Hence

$$
\begin{aligned}
f_k(\pi') - f_k(\pi^*) = \quad & (\pi'_{v_{l-1}} - \pi^*_{v_{l-1}}) \underbrace{\sum_{j=l}^{u} w_{v_j}}_{\geq 0^{\dagger}} + \underbrace{\sum_{j=l-1}^{u-2} \sum_{p=j+1}^{u-1} w_{v_p} s'_{a_j}}_{\substack{\geq 0^{\dagger} \text{ if } s' \neq s^{\alpha} \text{ in } f_k \\ =0 \text{ else}}} \\
& + \underbrace{w_{v_u}}_{\geq 0^{\dagger}} \left[\underbrace{\sum_{j=l-1}^{u-1} s'_{a_j}}_{\geq \alpha} - \underbrace{\sum_{j=l-1}^{u-1} s^{\alpha}_{a_j}}_{=\alpha} \right]
\end{aligned}
$$

where in all places marked with †, $\geq$ gets $>$ if we assume $w_u > 0$ for all $u \in V$. It remains to show $\pi'_{v_{l-1}} \geq \pi^*_{v_{l-1}}$. Each timetable π satisfies $\pi_{v_{j+1}} = \pi_{v_j} + L_{v_j} + s_{a_j}$, $j = 1, \ldots, |V| - 1$, and each robust timetable also satisfies (6.9), so we have

$$\begin{aligned}\pi_{v_{l-1}} &= \pi_{v_{k(\Delta+1)+1}} \\ &= \pi_{v_1} + \sum_{j=1}^{k(\Delta+1)} (L_{a_j} + s_{a_j}) \\ &= \pi_{v_1} + \sum_{j=1}^{k(\Delta+1)} L_{a_j} + \sum_{j=1}^{k(\Delta+1)} s_{a_j} \\ &\geq \pi_{v_1} + \sum_{j=1}^{k(\Delta+1)} L_{a_j} + k\alpha\end{aligned}$$

for each robust timetable π. As π^* satisfies (6.8), it satisfies (6.9) with equality, i.e.

$$\pi^*_{v_{l-1}} = \pi^*_{v_1} + \sum_{j=1}^{k(\Delta+1)} L_{a_j} + k\alpha,$$

so $\pi'_{v_{l-1}} \geq \pi^*_{v_{l-1}}$ which yields $f_k(\pi') \geq f_k(\pi^*)$ for each k. Hence, π^* is optimal. If we assume $w_u > 0$ for all $u \in V$, then $f_k(\pi') \gneqq f_k(\pi^*)$ for each summand f_k for which $s' \neq s^\alpha$ (at least one), hence $f(\pi') \gneqq f(\pi^*)$, so π^* is the unique optimal solution. □

Corollary 6.20. *If G is a linear graph, there exists a linear time algorithm that computes optimal solutions for $\mathcal{TT}^v_1$.*

6.3.2 Limited-Delay and Timetabling with Arc Weights

If we consider the recoverable-robust timetabling problem $\mathcal{TT}^a_1 = (TT^a, M, \mathbb{A}_\delta)$, the following results on algorithm Alg^+_s hold:

Lemma 6.21. *Let G be a tree and let $w_a > 0$ for at least one $a \in A$. If Alg^+_s is robust, its price of robustness is*

$$P_{\text{rob}}(\mathcal{TT}^a_1, \mathsf{Alg}^+_s) \leq 1 + \frac{s}{L_{\min}}.$$

Proof.

$$\begin{aligned} P_{\text{rob}}(\mathcal{TT}_1^a, \mathsf{Alg}_s^+) &= \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a(L_a^0 + s)}{\sum_{a=(u,v)\in A} w_a L_a^0} \\ &= 1 + s \cdot \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a}{\sum_{a=(u,v)\in A} w_a L_a^0} \\ &\leq 1 + s \cdot \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a}{\sum_{a=(u,v)\in A} w_a L_{\min}} \\ &\leq 1 + \frac{s}{L_{\min}}. \end{aligned}$$

□

Note that the generalization of Lemma 6.21 to the multi-stage case also holds, see Lemma 6.37.

Lemma 6.22. *Let $\delta \leq \frac{\alpha}{2}$. Then $\mathsf{Alg}_{\alpha-\delta}^+$ is robust. Furthermore, if G is a tree, its price of robustness is*

$$P_{\text{rob}}(\mathcal{TT}_1^a, \mathsf{Alg}_{\alpha-\delta}^+) \leq 1 + \frac{\alpha - \delta}{L_{\min}}.$$

Proof. Let π be a solution computed by $\mathsf{Alg}_{\alpha-\delta}^+$ and let x be the solution after the recovery phase. Assume that arc $(u, v) \in A$ is delayed by α. Let w_j, $j = 1, \ldots, l$, be the set of nodes directly connected to v by an arc $(v, w_j) \in A$, see Figure 6.4.

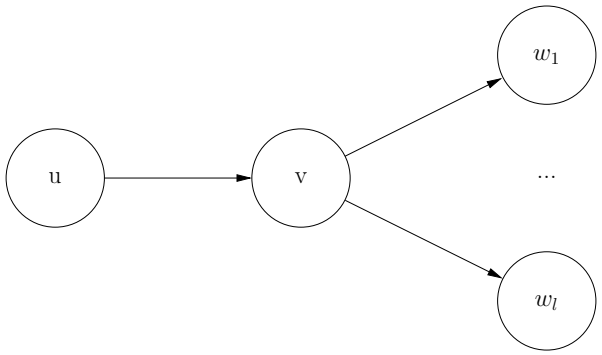

Figure 6.4: Illustration of the proof of Lemma 6.22.

We calculate the delays as

$$\begin{aligned} x_v - \pi_v &= \alpha - (\alpha - \delta) = \delta \\ x_{w_j} - \pi_{w_j} &\leq [\delta - (\alpha - \delta)]_+ = 0 \text{ for all } j = 1, \ldots, l, \end{aligned}$$

the latter using $\delta \leq \frac{\alpha}{2}$. Hence,

$$\sum_{u \in V} (x_u - \pi_u) = \delta,$$

i.e. π is robust. The price of robustness follows from Lemma 6.21. □

6.4 Multi-Stage Recoverable Robustness

Now we extend the single-stage model in order to deal with a sequence of $\sigma \geq 1$ modifications. The main motivation lies on the observation that in many applications, one is typically not facing only one disturbing event, but several disturbances $i^1, i^2, \ldots, i^\sigma$ may occur. This obviously is the case for the operational phase in public transportation where several different delays might occur one after another. For example, assume that we expect at most two disturbances i^1 and i^2. In this case, a robust solution for i^1 should be also recoverable against the next disturbance i^2. This means that under all solutions which are robust for i^1, we should choose one that is again robust against the next disturbance i^2 (if it exists). This example can be extended to more than two disturbances, see Figure 6.5 for an example where up to 3 disturbances are allowed.

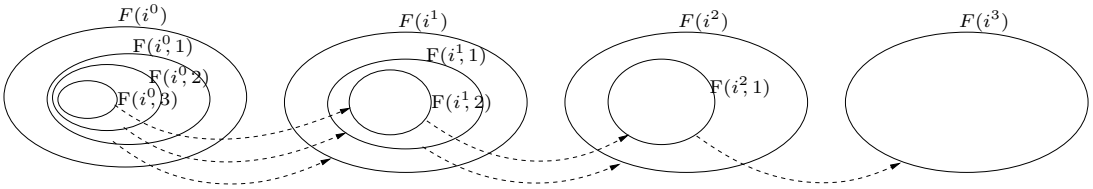

Figure 6.5: The set of solutions that are recoverable against 1, 2, and 3 disturbances. $\mathrm{F}(i, n)$ denotes the set of feasible solutions for a problem which has to be solved against n disturbances. Dotted arrows represent recovery algorithms.

To describe robustness, as for the single-stage case, the following concepts are necessary.

- $\sigma \in \mathbb{N}$ denotes the maximum number of expected modifications. In a practical scenario, several disruptions $i^1, i^2, \ldots, i^\sigma$ may occur. In this case, the task is to devise recovery algorithms that can recompute the solution for P after *each* disruption. The introduction of more than one disturbance extends the concept of recoverable robustness presented in Section 6.2 where only one single disturbance is taken into account.

- The modification function $M : I \to 2^I$. Note that M might also depend on other information, e.g. $M(i^k)$ may depend on data of instances i^0, i^1, ..., i^{k-1}.
- Again, $\mathbb{A}_{\text{rec}}$ denotes the class of recovery algorithms for P. Note that s^0 and i^1 define the minimal amount of information necessary to recompute the recovered solution. However, for specific cases, A_{rec} could require additional information. In general, when A_{rec} is used in the k-th stage, it could use everything that has been processed in the previous stages, i.e. $i^0, ..., i^{k-1}$ and $s^0, ..., s^{k-1}$.

The following definitions are extensions of Definition 6.1 and Definition 6.2 to the multi-stage case:

Definition 6.23. *A* multi-stage recoverable robustness problem *is given by a tuple* $(P, M, \mathbb{A}_{\text{rec}}, \sigma)$ *where* $(P, M, \mathbb{A}_{\text{rec}}) \in \text{RRP}$ *is a recoverable robustness problem and* $\sigma \in \mathbb{N}$. *The class* $\text{RRP}(\sigma)$ *contains all multi-stage recoverable robustness problems, i.e. all recoverable robustness problems that have to be solved against* $\sigma \geq 1$ *possible disruptions.*

Definition 6.24. *Let* $\sigma \in \mathbb{N}$ *and* $\mathcal{P} = (P, M, \mathbb{A}_{\text{rec}}, \sigma)$ *be an element of* $\text{RRP}(\sigma)$. *Given an instance* $i^0 \in I$ *of* P, s^0 *is a* feasible solution *for* i^0 *with respect to* $\mathcal{P}$ *if and only if the following relationship holds:*

$$\exists A_{\text{rec}} \in \mathbb{A}_{\text{rec}} : \quad s^0 \in F(i^0) \tag{6.10}$$

$$s^k := A_{\text{rec}}(s^{k-1}, i^k) \in F(i^k) \quad \forall i^k \in M(i^{k-1}),\ \forall k \in \{1, \ldots, \sigma\}. \tag{6.11}$$

This definition ensures that for each stage k, for any possible modification $i^k \in M(i^{k-1})$ and for any feasible solution s^{k-1} computed in the previous stage, the output s^k of algorithm A_{rec} is a feasible solution for i^k with respect to P. If it is clear to which problem P, M and $\mathbb{A}_{\text{rec}}$ we refer to, we also say in short that s^0 is *feasible for i with respect to σ recoveries*. As in the single-stage case, $F_{\mathcal{P}}(i)$ is considered as the set of *robust solutions* for i with respect to the original problem P.

Note that $\text{RRP}(1) = \text{RRP}$. Hence, each problem in $\text{RRP}(1)$ is called a *single-stage recoverable robustness problem* and each problem in $\text{RRP}(\sigma)$, $\sigma > 1$, is called a *multi-stage recoverable robustness problem.*

Using the definition of a feasible solution, the robust algorithm that is used to compute the initial solution s^0 for the initial (undisturbed) instance i^0 is defined in the following definition similar to Definition 6.3 for the single-stage case:

Definition 6.25. *Given a multi-stage recoverable robustness problem* $\mathcal{P} \in \text{RRP}(\sigma)$, *a* robust algorithm *for* $\mathcal{P}$ *is any algorithm* $A_{\text{rob}} : I \to S$ *such that for each* $i \in I$, $A_{\text{rob}}(i)$ *is feasible for i with respect to* $\mathcal{P}$, *i.e. such that* A_{rob} *outputs a solution that can be recovered against* σ *disturbances.*

Note that A_{rob} as in the definition above provides the solution s^0 as defined in Definition 6.24. In the case of strict robustness, a robust algorithm A_{rob} for $\mathcal{P}$ must provide a solution s^0 for i^0 such that for each possible modification $i^k \in M(i^{k-1})$, we have $s^0 \in F_{\mathcal{P}}(i^k)$ for all $k \in \{1, \ldots, \sigma\}$. The meaning is the following: If A_{rec} has no recovery capability, then A_{rob} has to find solutions that "absorb" *any* possible sequence of disturbances.

Analogously to the single-stage case, the price of robustness can also be defined for multi-stage recovery algorithms. Definitions 6.26-6.28 are generalizations of Definitions 6.4-6.6 to the multi-stage case.

For every instance $i \in I$ of the initial problem P, the price of robustness of the robust algorithm A_{rob} is given by the maximum ratio between the cost of the solution provided by A_{rob} and the cost of an optimal (non-robust) solution. The following definition differs from the corresponding definition for the single-stage case only in the problem $\mathcal{P}$ (it now belongs to the larger class $\text{RRP}(\sigma)$ instead of RRP).

Definition 6.26. *The* price of robustness *of a robust algorithm A_{rob} for a recoverable robustness problem $\mathcal{P} \in \text{RRP}(\sigma)$ is given by*

$$P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) := \max_{i \in I} \left\{ \frac{f(A_{\text{rob}}(i))}{\min\{f(x) : x \in F(i)\}} \right\}.$$

The price of robustness of a recoverable robustness problem $\mathcal{P} \in \text{RRP}(\sigma)$ is then given by the minimum price of robustness among all possible robust algorithms for this problem. Formally:

Definition 6.27. *The* price of robustness *of a multi-stage recoverable robustness problem $\mathcal{P} \in \text{RRP}(\sigma)$ is given by*

$$P_{\text{rob}}(\mathcal{P}) = \min\{P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) : A_{\text{rob}} \text{ is a robust algorithm for } \mathcal{P}\}.$$

If there are different robust algorithms for a recoverable robustness problem, we want to identify the "best" one:

Definition 6.28. *Let $\mathcal{P} \in \text{RRP}(\sigma)$ and let A_{rob} be a robust algorithm for $\mathcal{P}$. Then,*

- *A_{rob} is called* $\mathcal{P}$-optimal *if $P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) = P_{\text{rob}}(\mathcal{P})$;*
- *A_{rob} is called* exact *if $P_{\text{rob}}(\mathcal{P}, A_{\text{rob}}) = 1$.*

We call a solution computed by an optimal algorithm $\mathcal{P}$-optimal*, while a solution computed by an exact algorithm is called* exact*.*

Note that each exact algorithm is optimal.

We now state a simple observation concerning the price of robustness.

Lemma 6.29. *For fixed P, M, and $\mathbb{A}_{\text{rec}}$, consider a family of recoverable robustness problems $\mathcal{P}_\sigma = (P, M, \mathbb{A}_{\text{rec}}, \sigma) \in \text{RRP}(\sigma)$ for different values of σ, i.e. these problems vary in the expected number of recoveries only. For $\sigma_1 < \sigma_2$, we have*

- *$F_{\mathcal{P}_{\sigma_2}}(i) \subseteq F_{\mathcal{P}_{\sigma_1}}(i)$ for all instances $i \in I$,*
- *$P_{\text{rob}}(\mathcal{P}_{\sigma_1}) \leq P_{\text{rob}}(\mathcal{P}_{\sigma_2})$, i.e. the price of robustness grows in the number of expected recoveries.*

Proof. Let A_{rob} be a robust algorithm for $\mathcal{P}_{\sigma_2}$. Let $i \in I$ and $s \in F_{\mathcal{P}_{\sigma_2}}(i)$. By Definition 6.24, there exists a recovery algorithm $A_{\text{rec}} \in \mathbb{A}_{\text{rec}}$ such that (6.11) holds for all $k = 1, \ldots, \sigma_2$, hence also for all $k = 1, \ldots, \sigma_1 < \sigma_2$. This implies $s \in F_{\mathcal{P}_{\sigma_1}}(i)$.

The second statement is a straightforward consequence of the first one. □

6.5 Multi-Stage Recoverable-Robust Timetabling

We now apply the concept of multi-stage recoverable robustness to a simple version of the timetabling problem. In fact, we generalize the recoverable robustness problems $\mathcal{TT}_1$, $\mathcal{TT}_1^v$ and $\mathcal{TT}_1^a$ from Section 6.3 to multiple-stage recoverable robustness problems $\mathcal{TT}_\sigma$, $\mathcal{TT}_\sigma^v$ and $\mathcal{TT}_\sigma^a$. To this end, we formalize the modifications function M and the set $\mathbb{A}_{\text{rec}}$ of recovery algorithms which we need to define a multi-stage recoverable robustness problem $\mathcal{P} = (P, M, \mathbb{A}_{\text{rec}}, \sigma) \in \text{RRP}(\sigma)$ as follows:

- As in Section 6.3, the modification function for an instance $i^{k-1} = (G, w, L^{k-1})$ and a constant $\alpha \in \mathbb{R}^{>0}$ is

  $$M(i^{k-1}) = \left\{ (G, w, L^k) : \exists \overline{a} \in \mathcal{A} : L^0_{\overline{a}} \leq L^k_{\overline{a}} \leq L^0_{\overline{a}} + \alpha \text{ and } L^k_a = L^{k-1}_a \ \forall a \neq \overline{a} \right\},$$

  i.e. we allow one additional delay (its size bounded by α) in every stage k and, if the additional delay concerns an already delayed activity, the total delay cannot exceed the value of α.

- The class $\mathbb{A}_{\text{rec}}$ of recovery algorithms is based on the same two limitations as in the single-stage case, namely classes $\mathbb{A}_\Delta$ and $\mathbb{A}_\delta$. In particular, we require that a solution x^k computed by a recovery algorithm A_{rec} satisfies $d(x^k, \pi) \leq \Delta$ (δ) when $A_{\text{rec}} \in \mathbb{A}_\Delta$ ($\mathbb{A}_\delta$).

As in the single-stage case, using those definitions, we can turn the initial problem TT into a multi-stage recoverable robustness problem $\mathcal{TT}_\sigma = (TT, M, \mathbb{A}_{\text{rec}}, \sigma) \in \text{RRP}(\sigma)$. Special cases of $\mathcal{TT}_\sigma$ can be obtained by replacing TT with TT^a or TT^v and choosing a class of recovery algorithms. Among the four possible combinations of initial optimization problem and class of recovery algorithm, we focus on the same two recoverable robustness problems as in Section 6.3:

Robust timetabling with nonnegative node weights and limited-events. In the first recoverable robustness problem, we use the special timetabling problem TT^v with nonnegative node weights and the class $\mathbb{A}_\Delta$ of recovery algorithms. The resulting recoverable robustness problem is

$$\mathcal{TT}^v_\sigma := (TT^v, M, \mathbb{A}_\Delta, \sigma).$$

Robust timetabling with arc weights and limited-delay. In this case, we consider the special timetabling problem TT^a with objective function f_{arcs} together with the class $\mathbb{A}_\delta$ of recovery algorithms. The corresponding recoverable robustness problem hence is

$$\mathcal{TT}^a_\sigma := (TT^a, M, \mathbb{A}_\delta, \sigma).$$

Solving these problems requires to define robust algorithms as stated in Definition 6.25. For instance, let us assume that we want to solve $\mathcal{TT}^v_\sigma$. In this case, we need a robust algorithm A_{rob} such that, for a given instance $i^0 = (G, w, L^0)$ of TT^v, it computes a timetable π satisfying

$$\pi_v - \pi_u \geq L^0_a \quad \forall a = (u, v) \in A$$

(this defines a *feasible* timetable) and such that for all modifications $(G, w, L^k) \in M(i^{k-1})$ of i^{k-1}, $k \in \{1, \ldots, \sigma\}$, there exists a disposition timetable x^k with

$$\begin{aligned} x^k_v - x^k_u &\geq L^k_a && \forall a = (u, v) \in A \\ x^k_u &\geq \pi_u && \forall u \in V \\ d(x^k, \pi) &\leq \Delta \end{aligned} \tag{6.12}$$

(this defines a *robust* timetable). In case $\mathcal{TT}^a_\sigma$ is considered instead of $\mathcal{TT}^v_\sigma$, the above constraints are still valid, except for (6.12) where Δ has to be replaced by δ.

First, we show the connection between recoverable-robust timetabling and multi-stage recoverable-robust timetabling. From Lemma 6.29, we know $F_{\mathcal{TT}^v_1}(i^0) \supseteq F_{\mathcal{TT}^v_\sigma}(i^0)$. In general, $F_{\mathcal{TT}^v_1}(i^0) \nsubseteq F_{\mathcal{TT}^v_\sigma}(i^0)$; however, in the case of strict robustness (i.e. $\Delta = 0$), the following lemma implies $F_{\mathcal{TT}^v_1}(i^0) = F_{\mathcal{TT}^v_\sigma}(i^0)$.

Lemma 6.30. *Let $\mathcal{TT}^v_1$ and $\mathcal{TT}^v_\sigma$ defined with $\Delta = 0$. If A_{rob} is a robust algorithm for $\mathcal{TT}^v_1$, then A_{rob} is robust for $\mathcal{TT}^v_\sigma$.*

Proof. Let A_{rob} be a robust algorithm for $\mathcal{TT}^v_1$, and let $\pi = A_{\text{rob}}(i^0)$. According to Lemma 6.9, π assigns a slack of at least α to each activity. As in the modification function M, the maximal delay of each activity is at most α, A_{rob} is also a robust algorithm for $\mathcal{TT}^v_\sigma$. □

Corollary 6.31. *Lemma 6.30 also holds for $\mathcal{TT}^a_1$ and $\mathcal{TT}^a_\sigma$ if $\delta = 0$.*

6.5.1 Limited-Events and Timetabling with Node Weights

In this section, we consider the recoverable robustness problem $\mathcal{TT}^v_\sigma = (TT^v, M, \mathbb{A}_\Delta, \sigma)$. As in the single-stage case, each timetable is robust if Δ is "large enough", compared to the size of the event-activity network, i.e. if $\Delta \geq |V| - 1$ (see Lemma 6.13). Hence, in the following, we always assume $\Delta \leq |V| - 2$.

If $\sigma > \Delta$, a straightforward consequence of the considered modification function M is that we need strict robustness to get a robust solution:

Lemma 6.32. *If $\sigma > \Delta$, then a timetable is robust if and only if the slack s satisfies $s_a \geq \alpha$ for each $a \in A$. In this case, we have strict robustness.*

In the following, we restrict our analysis to linear graphs.

Linear graphs.

We now suggest a robust algorithm for arbitrary σ for a linear graph. It is based on algorithm Alg^+_s and assigns the same slack

$$s^* = \min\left\{\alpha, \frac{\sigma\alpha}{\Delta + 1}\right\} \tag{6.13}$$

to each arc. In Theorem 6.35, we show that $\mathsf{Alg}^+_{s^*}$ is robust for $\mathcal{TT}^v_\sigma$ and that it is optimal compared to all other robust algorithms that add an equal slack s to all arcs. We need the following two results for the proof:

Lemma 6.33. *If $s_a < \alpha$ for all arcs $a \in A$, then the number of nodes affected by a single delay of $\sigma\alpha$ on arc $a_j = (v_j, v_{j+1})$ is equal to the number of nodes affected by σ single delays of α on the σ consecutive arcs $a_{j+k} = (v_{j+k}, v_{j+k+1})$, $k = 0, \ldots, \sigma - 1$.*

Proof. Let $s_a < \alpha$ for all arcs $a \in A$. If, on the one hand, a_j is delayed by $\sigma\alpha$, then v_{j+1} has a delay of $\sigma\alpha - s_{a_j}$, v_{j+2} has a delay of $\sigma\alpha - s_{a_j} - s_{a_{j+1}}$ and so on, and $v_{j+\sigma}$ has a delay of $\sigma\alpha - \sum_{k=0}^{\sigma-1} s_{a_{j+k}}$. If, on the other hand, $a_j, \dots, a_{j+\sigma-1}$ are delayed by α, then v_{j+1} has a delay of $\alpha - s_{a_j}$, v_{j+2} has a delay of $2\alpha - s_{a_j} - s_{a_{j+1}}$ and so on, and $v_{j+\sigma}$ has a delay of $\sigma\alpha - \sum_{k=0}^{\sigma-1} s_{a_{j+k}}$. As $s_a < \alpha$ for all arcs $a \in A$, all these delays are positive. Hence in both cases, the nodes $v_{j+1}, \dots, v_{j+\sigma}$ are affected, and as the delay of $v_{j+\sigma}$ is the same in both cases, the total number of subsequent affected nodes is the same, too. □

Lemma 6.34. *If all arcs $a \in A$ have the same slack $s_a = s$, then the number of nodes affected by σ delays of α on σ consecutive arcs is always greater than or equal to the number of nodes affected by σ delays of α on σ non-consecutive arcs.*

Proof. W.l.o.g., we may assume $s < \alpha$ (otherwise no node at all is affected by a delay). If a delay of α occurs, then we need $\lceil \frac{\alpha}{s} \rceil$ arcs with slack s to let the delay vanish. Hence, a single delay affects $\lceil \frac{\alpha}{s} \rceil - 1$ nodes, and thus, σ delays of α affect at least $\sigma \left(\lceil \frac{\alpha}{s} \rceil - 1 \right)$ nodes. If the sets of nodes affected by each delay are disjoint, then each delay exactly affects $\lceil \frac{\alpha}{s} \rceil - 1$ nodes. It remains to show that the total of affected nodes grows if the sets of affected nodes of each delay are not disjoint.

Let $\sigma = 2$ and assume that the first delay occurs on arc $a_j = (v_j, v_{j+1})$ and affects the k nodes $v_{j+1}, v_{j+2}, \dots, v_{j+k}$. The last affected node is node v_{j+k} with a delay of $(x_{j+k} - \pi_{j+k}) \in (0, s]$. Now, assume that the second delay occurs on some arc $a_{j'} = (v_{j'}, v_{j'+1})$, $j' \in \{j+1, \dots, j+k\}$. As the whole slack of arcs a_i, $i = j, \dots, j+k-1$, and a part of the slack of arc a_{j+k} is already used to reduce the first delay, the second delay also affects $v_{j'}, \dots, v_{j+k}$ that are already affected by the first delay (this does not change the total number of affected nodes). Starting with arc a_{j+k}, the second delay can be reduced on each subsequent arc until it vanishes. On arc a_{j+k}, it can be reduced by the remaining slack $(s - (x_{j+k} - \pi_{j+k})) \in [0, s)$; on all subsequent arcs, it can be reduced by the full slack s. If in this example $y_{j+k} > \frac{s}{2}$, then one easily sees that in this case, the second delay needs a total of $\lceil \frac{\alpha}{s} \rceil + 1$ arcs to vanish instead of $\lceil \frac{\alpha}{s} \rceil$ as it is the case when the sets of affected nodes of each delay are disjoint; thus, both delays together influence $2 \left(\lceil \frac{\alpha}{s} \rceil - 1 \right) + 1$ nodes.

The idea of the proof for $\sigma = 2$ can be extended arbitrary σ. □

Now, we can prove that $\mathsf{Alg}^+_{s^*}$ is robust for $\mathcal{TT}^v_\sigma$ and that it is optimal compared to all robust algorithms that add an equal slack s to all arcs:

Theorem 6.35. *Let G be a linear graph and let s^* be defined as in* (6.13)*. Assume that we add the same slack time s to all arcs. Then Alg^+_s is a robust algorithm for $\mathcal{TT}^v_\sigma$*

if and only if $s \geq s^*$. If $s > s^*$ and $|V| > 1$, then $f(\mathsf{Alg}_s^+(i)) > f(\mathsf{Alg}_{s^*}^+(i))$ if $w_u > 0$ for at least one node $u \in V \setminus \{v_1\}$.

Proof. If $\sigma > \Delta$, according to Lemma 6.32, $s_a = \alpha$ for all $a \in A$ is the optimal solution; in this case, due to (6.13), we have $s^* = \alpha$ and the theorem holds. So in the following, let $\sigma \leq \Delta$ which implies $s^* < \alpha$. At first we show that Alg_s^+ is a robust algorithm for $\mathcal{TT}_\sigma^v$ if $s \geq s^*$. We look at the worst case that the first delay of α occurs on arc a_j, the second delay of α occurs on arc a_{j+1} and so on; in the proof of Lemma 6.34, we explained why this is the worst case. For the sake of simplicity, we may assume that we only have one large delay of $\sigma\alpha$ on arc a_j, see Lemma 6.33. Given a slack s, $\lceil \frac{\sigma\alpha}{s} \rceil$ arcs are needed to reduce this delay until it vanishes. On the last of these arcs, the delay is reduced to 0, so the node at the end of this last arc has no delay. Hence, the number of nodes influenced by the delays is $d(x, \pi) = \lceil \frac{\sigma\alpha}{s} \rceil - 1$. For $s \geq s^*$, by using (6.13), we have $d(x, \pi) \leq \Delta$. If $s < s^*$, then $d(x, \pi) > \Delta$. Hence, $\mathsf{Alg}_{s^*}^+$ is robust if and only if $s \geq s^*$.

For the second part, let $s > s^*$. By π^* (π), we denote the timetable with slack s^* (s). Then, according to the definition of CPM, $\pi_{v_1} = \pi_{v_1}^* = 0$ and $\pi_{v_j} > \pi_{v_j}^*$ for all nodes $v_j \in V \setminus \{v_1\}$. This yields $f(\pi) > f(\pi^*)$ if $|V| > 1$. □

Theorem 6.36. *Let G be a linear graph, and let s^* be defined as in* (6.13)*. If, for each $(G, w, L^0) \in I$, $w_a > 0$ for at least one $a \in A$, then the following holds:*

$$P_{\text{rob}}(\mathcal{TT}_\sigma^v, \mathsf{Alg}_{s^*}^+) \leq 1 + \frac{s^*}{L_{\min}}.$$

Proof. Using the definition of CPM, we have $\pi_{v_1} = 0$ and $\pi_{v_j} = \sum_{k=1}^{j-1}(L_{a_k} + s_{a_k})$, $j = 2, \ldots, n$. This yields

$$\begin{aligned}
P_{\text{rob}}\left(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_{s^*}^+\right) &= \max_{i=(G,w,L^0)\in I} \frac{f\left(\mathsf{Alg}_{s^*}^+(i)\right)}{\min\{f(\pi) : \pi \in F(i)\}} \\
&= \max_{i=(G,w,L^0)\in I} \frac{\sum_{j=2}^{n}\sum_{k=1}^{j-1} w_{v_j}(L_{a_k} + s^*)}{\sum_{j=2}^{n}\sum_{k=1}^{j-1} w_{v_j} L_{a_k}} \\
&= 1 + s^* \max_{i=(G,w,L^0)\in I} \frac{\sum_{j=2}^{n}\sum_{k=1}^{j-1} w_{v_j}}{\sum_{j=2}^{n}\sum_{k=1}^{j-1} w_{v_j} L_{a_k}} \\
&\leq 1 + \frac{s^*}{L_{\min}}
\end{aligned}$$

using $L_a \geq L_{\min}$ for all $a \in A$. □

6.5.2 Limited-Delay and Timetabling with Arc Weights

We complete our analysis of recoverable-robust timetabling with an investigation of the recoverable robustness problem $\mathcal{TT}^a_\sigma = (TT^a, M, \mathbb{A}_\delta, \sigma)$. Again, our strategy to make a timetable robust against delays is to add a slack time s_a to all of the arcs and to apply algorithm Alg^+_s afterwards. In the following we consider algorithm Alg^+_s with $s_a = s$ for all $a \in A$, i.e. we add the same amount of slack time to all activities.

We first investigate how much we loose if we use Alg^+_s instead of an algorithm that computes the optimal (but not robust) solution of TT^a, i.e. without the additional slack s. Again, let $L_{\min}$ be the minimum value assigned by the function L with respect to all the possible instances of $\mathcal{TT}^a_\sigma$.

The following lemma is a generalization of Lemma 6.21 from the single-stage case; the proof is exactly the same:

Lemma 6.37. *Let G be a tree and let $w_a > 0$ for at least one $a \in A$. If Alg^+_s is robust, its price of robustness is*

$$P_{\text{rob}}(\mathcal{TT}^a_\sigma, \mathsf{Alg}^+_s) \leq 1 + \frac{s}{L_{\min}}.$$

Now we discuss how much slack time s is needed to guarantee robustness of Alg^+_s.

Strict robustness.

Our first result deals with strict robustness, i.e. if $\delta = 0$. In this case we have to make sure that any delay can be compensated by the slack time on the corresponding edge. Since L^k_a never differs from L^0_a by more than α in any scenario, it suffices to add an additional slack of α to each L^0_a for all $a \in A$. Then the resulting disposition timetable in each step equals the original timetable π, i.e. a recovery step is in fact not necessary.

Lemma 6.38. *The algorithm Alg^+_α is strictly robust (i.e. it is robust for the case $\delta = 0$) for any graph G. Furthermore, if G is a tree, its price of robustness is*

$$P_{\text{rob}}(\mathcal{TT}^a_\sigma, \mathsf{Alg}^+_\alpha) \leq 1 + \frac{\alpha}{L_{\min}}.$$

Proof. The robustness of Alg^+_α is clear. The price of robustness follows from Lemma 6.37. □

Next, we turn our attention to the case $\delta > 0$.

Linear graphs.

Simplifying the network to a linear graph allows us to drop the restrictions on δ from Lemma 6.38 and thus yields the following result for algorithm Alg_s^+:

Theorem 6.39. *Let G be a linear graph. Then Alg_s^+ is robust for $\mathcal{TT}_\sigma^a$ if and only if*

$$s \geq s^* := \frac{2\sigma\alpha\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil + \sigma\right) - \sigma\alpha(\sigma+1) - 2\delta}{\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil + \sigma\right)\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil + \sigma - 1\right)}.$$

Proof. If $s \geq \alpha$, we have strict robustness and the theorem holds as $s^* \leq \alpha$. So in the following, we may assume $s < \alpha$. To prove robustness, we analyze the worst case scenario. By an argument similar to Lemma 6.34, in the worst case, all delays occur on consecutive arcs, so w.l.o.g. we assume that we have σ delays of α on the σ consecutive arcs $a_1, \ldots, a_\sigma$. To simplify the subsequent calculations, we instead consider the case of one large delay of $\sigma\alpha$ on arc a_1. To justify this approach, we compare the effects of

(a) one delay of $\sigma\alpha$ on arc a_1, and

(b) σ delays of α on the σ consecutive arcs $a_1, \ldots, a_\sigma$.

In case (a), v_1 has a delay of $d_1^{(a)} = \sigma\alpha - s$, v_2 has a delay of $d_2^{(a)} = \sigma\alpha - 2s$ and so on, and v_σ has a delay of $d_\sigma^{(a)} = \sigma\alpha - \sigma s$. In case (b), v_1 has a delay of $d_1^{(b)} = \alpha - s = \sigma\alpha - s - (\sigma-1)\alpha$, v_2 has a delay of $d_2^{(b)} = 2\alpha - 2s = \sigma\alpha - 2s - (\sigma-2)\alpha$ and so on, and v_σ has a delay of $d_\sigma^{(b)} = \sigma\alpha - \sigma s$. As $s < \alpha$, all those delays are positive. Hence, we have

$$\sum_{k=1}^{\sigma} d_k^{(b)} = \sum_{k=1}^{\sigma} d_k^{(a)} - \sum_{k=1}^{\sigma-1} k\alpha,$$

and starting from node v_σ, all subsequent delays are the same. So instead of calculating the sum of all delays for case (b), we can focus on case (a) and subtract $\sum_{k=1}^{\sigma-1} k\alpha$ afterwards.

For $s = 0$, it is clear that we do not have robustness, so we assume $s > 0$ in the following. We consider an instance $i = (G, w, L^0) \in I$ with $|V| \geq \left\lfloor\frac{\sigma\alpha}{s}\right\rfloor + 1$. Then

$$\begin{aligned}
\sum_{u\in V}(x_u - \pi_u) &= \sum_{k=1}^{\left\lceil\frac{\sigma\alpha}{s}\right\rceil - 1}(\sigma\alpha - ks) - \sum_{k=1}^{\sigma-1} k\alpha \\
&= \left(\left\lceil\frac{\sigma\alpha}{s}\right\rceil - 1\right)\sigma\alpha - s\frac{\left(\left\lceil\frac{\sigma\alpha}{s}\right\rceil - 1\right)\left\lceil\frac{\sigma\alpha}{s}\right\rceil}{2} - \alpha\frac{(\sigma-1)\sigma}{2} \\
&= \frac{1}{2}s\left\lceil\frac{\sigma\alpha}{s}\right\rceil - \frac{1}{2}s\left(\left\lceil\frac{\sigma\alpha}{s}\right\rceil\right)^2 + \sigma\alpha\left\lceil\frac{\sigma\alpha}{s}\right\rceil - \frac{1}{2}\sigma\alpha(\sigma+1).
\end{aligned}$$

Plugging in the formula for s^* shows (after some calculations) the result. □

Corollary 6.40. *It holds that* $s^* \geq \frac{\sigma^2\alpha^2}{2\delta+\sigma^2\alpha}$ *where equality holds if* $\frac{s\delta}{\sigma\alpha}$ *is integer.*

Proof. One can compute that $s^* \geq \frac{\sigma^2\alpha^2}{2\delta+\sigma^2\alpha}$ if and only if

$$\underbrace{\left\lceil \frac{2\delta}{\sigma\alpha} \right\rceil \sigma\alpha}_{:=A} \underbrace{\left(4\delta + \sigma\alpha - \sigma\alpha \left\lceil \frac{2\delta}{\sigma\alpha} \right\rceil\right)}_{:=B} \geq \underbrace{2\delta}_{:=C} \underbrace{(2\delta + \sigma\alpha)}_{:=D}.$$

For the latter expression note that $A, B, C, D \geq 0$ and that $A + B = C + D$ and that $A - B \leq D - C$. Hence $AB \geq CD$ and the lower bound is established (given two rectangles with the same perimeter, the one with the smaller difference between width and length has the larger area). Plugging in s^* in the case that $\frac{s\delta}{\sigma\alpha}$ is integer shows (after some calculations) that equality holds. □

The price of robustness of algorithm $\mathsf{Alg}^+_{s^*}$ can finally be written down.

Corollary 6.41. *Let* G *be a linear graph. Then* $P_{\text{rob}}(\mathcal{TT}^a_\sigma, \mathsf{Alg}^+_{s^*}) = 1 + s^*$ *where* s^* *is the minimal slack time from Theorem 6.39.*

Note that for a concrete scenario a slack smaller than s^* might also give a robust timetable. This might happen for example if no two source-delayed arcs follow each other or if the size of the network is limited such that at least one node $u \in V$ with a delay of $(x_u - \pi_u) > s^*$ has no outgoing arc. However, we are not interested in one special scenario, but in all possible scenarios from the set of admissible scenarios.

We also remark that this is a discussion about the price of robustness of Alg^+_s only. The question if there exists an approach which does better in the worst case is still open. But note that it need not be optimal to add the same slack s to all arcs when the weights w_a are different from each other. This can be seen in the following example.

Example 6.42. *Consider a linear graph with edges* $A = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$*, weights* $w = (1, 100, 1)$ *and lower bounds* $L^0 = (1, 1, 1)$*, see Figure 6.6 for an illustration.*

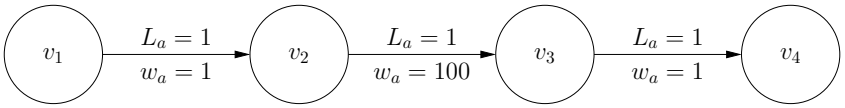

Figure 6.6: The DAG for Example 6.42.

Let $\alpha = 4$*,* $\delta = 5$ *and* $\sigma = 1$*. If we add the same slack to all arcs, we need at least a slack of 2 to achieve robustness. With* $s = (2, 2, 2)$*, we have*

$$\sum_{a=(u,v)\in A} w_a(\pi_v - \pi_u) \;=\; \sum_{a=(u,v)\in A} w_a(L_a^0 + s) \;=\; 306$$

in a time-minimal timetable. If, by contrast, we allow different slacks on the arcs and set $s = (2, 0, 3)$*, we get a robust timetable with*

$$\sum_{a=(u,v)\in A} w_a(\pi_v - \pi_u) = 107.$$

6.6 Summary

The results from Section 6.2-6.5 show how the algorithmic performances are affected by the variation on both the recovery capabilities (i.e. the class $\mathbb{A}_{\text{rec}}$) and the modification function M. As theoretically expected, the less restrictive the available recovery capabilities are, the smaller the price of robustness of an algorithm is. Vice versa, the larger the set of modification is, the larger the price of robustness of an algorithm is. Both observations can be seen in Table 6.8 and Table 6.9 where the price of robustness decreases with Δ (or δ, depending on the class $\mathbb{A}_{\text{rec}}$) and increases with α.

In conclusion, the price of robustness can be exactly calculated in special cases only; in many cases, an approximation is possible. Approximations are also needed when computing the optimal robust solution is hard. This situation occurs in some cases, as reported in Table 6.7.

Depending on the concrete restrictions on the recovery algorithms, the price of robustness might be very different. In Figure 6.10, we give the price of robustness for a linear graph if we either restrict the number of nodes being affected by a delay (see Theorem 6.36) or if we restrict the allowed deviation from the original timetable (see Corollary 6.41). It becomes clear that the price of robustness does not grow as rapidly as one might have expected, so further research in this direction seems to be promising also for real-world problems.

Graph	σ	Δ	Complexity	Result in
arbitrary	1	≥ 3	NP-hard	[CDSS08]
linear	any	any	linear	Corollary 6.20

Table 6.7: Computational complexity of calculating an optimal robust solution of $\mathcal{TT}^v_\sigma$.

Graph	σ	Δ	P_{rob}	Result in
DAG	1	any	$1 + \frac{\alpha}{L_{\min}}$	Lemma 6.15
linear	1	any	$P_{\text{rob}}(\mathcal{TT}^v_\sigma)$	Theorem 6.19
linear	any	any	$1 + \frac{1}{L_{\min}} \min\left\{\alpha, \frac{\sigma\alpha}{\Delta+1}\right\}$	Theorem 6.36

Table 6.8: Upper bounds for $P_{\text{rob}}(\mathcal{TT}^v_\sigma)$ in some cases. The algorithm considered in Theorem 6.19 is optimal, but its price of robustness has not been computed.

Graph	σ	δ	P_{rob}	Result in
tree	1	$\delta \leq \frac{\alpha}{2}$	$1 + \frac{\alpha-\delta}{L_{\min}}$	Lemma 6.22
tree	any	any	$1 + \frac{\alpha}{L_{\min}}$	Lemma 6.38
linear	any	any	$1 + \frac{2\sigma\alpha\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil+\sigma\right)-\sigma\alpha(\sigma+1)-2\delta}{\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil+\sigma\right)\left(\left\lceil\frac{2\delta}{\sigma\alpha}\right\rceil+\sigma-1\right)}$	Corollary 6.41

Table 6.9: Upper bounds for $P_{\text{rob}}(\mathcal{TT}^a_\sigma)$ in some cases.

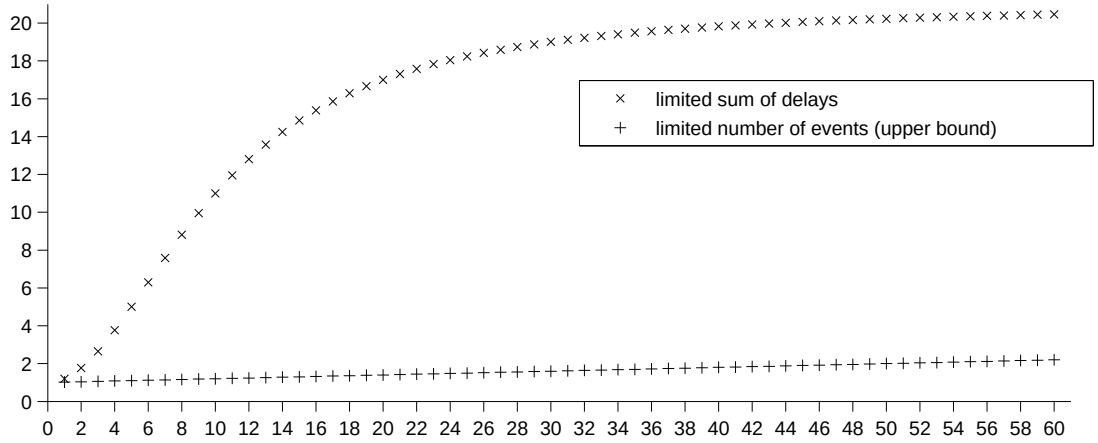

Figure 6.10: The price of robustness of algorithm $\mathsf{Alg}^{+}_{s^*}$ on a linear graph with $\alpha = 20$ and $\Delta = \delta = 1000$ as a function of σ with number of events as limitation (Theorem 6.36) and with sum of delays as limitation (Corollary 6.41).

Chapter 7

Discussion and Outlook

We conclude this thesis with a summary of the main results, a discussion of open problems, and an outlook on further research and possible extensions. We start with the summary in Section 7.1 and discuss open questions in Section 7.2. In Section 7.3, we present a programming framework that allows to numerically analyze the impact of different planning stages in public transportation on subsequent stages that is an important tool for further research; parts of it are based on results presented in this thesis. In Section 7.4, we conclude with an overview of possible extensions of our model.

7.1 Summary

In this work, we analyzed the delay management problem with capacity constraints. Based on an analysis of the headway activities where we showed that backward headways never carry over a delay to a *punctual* event, we were able to extend some results from the uncapacitated delay management problem to the capacitated case, especially concerning the never-meet property and related results. As a consequence, we were able to identify cases in which the objective of the integer programming formulation coincides with the exact sum of all delays of all passengers at their final destinations (while in all other cases, the objective is only an upper bound on this sum). While the never-meet property for uncapacitated delay management is "almost" satisfied in many real-world scenarios, the event-activity network with headway activities is much more dense, causing significantly more violations of the never-meet property. Based on the analysis of the headway activities and on results concerning the IP, we suggested different reduction techniques which can significantly reduce the size of an input instance. The numerical results from our case study, based on real-world data, show the efficiency of the suggested preprocessing procedures.

As the problem is NP-hard even in very special cases, applying the suggested reduction techniques might not be sufficient to be able to solve a large-scale real-world instance in a reasonable amount of time. Hence we suggested two classes of heuristic solution approaches and proved worst-case error bounds. As we have shown, the worst-case relative error of various classes of heuristics is arbitrarily large; the results are not limited to the actual solution procedures suggested in this thesis. We evaluated our solution procedures numerically on the same real-world data set that we used for testing the reduction techniques. It turned out that the theoretical results on the relative error are only worst-case results and that *in average*, the relative error of our heuristic approaches is rather limited. However, the relative error of different heuristics highly depends on the distribution of the source delays. By combining different heuristics, we derived a solution approach that is well suited for different distributions of source delays and that has a significantly lower average relative error than a single heuristic.

We also suggested how to integrate rolling stock circulations and delay management and analyzed the resulting problem. We extended results from capacitated delay management to the integrated problem, showed that it is NP-hard even in special cases, analyzed a polynomially solvable case, and suggested a generic solution framework.

In the last part of this work, we showed how to use the concept of recoverable robustness for computing delay resistant timetables. The resulting recoverable robustness problems are NP-hard, so we focused on analyzing special cases.

Yet, there are still open questions and room for further research as we show in the next section.

7.2 Open Questions and Further Work

In Section 4.1, we provided upper bounds on the relative error of Frfs-Fix and Frfs, using the input data of the actual instance. So far it is not clear whether similar bounds can be proven for Fsfs, Fsfs-Fix, and Priority-Repair. As we have shown in Chapter 4, such bounds definitely have to depend on the structure of the actual input instance. If a low worst-case relative error is required, the set of admissible input instances has to be very limited.

In Chapters 3 and 4, we numerically evaluated the reduction techniques and heuristic solution approaches using a real-world data set. It would be interesting to check whether the results also hold for different data sets with other network topologies and other timetables. Unfortunately, we did not have access to other real-world data. Once other data sets are avaliable, they can be compared easily by using the framework presented in Section 7.3.

Throughout our analysis of the delay management problem in Chapters 2-5, we focused on the model where we assumed all delays to be known. The resulting problem already is NP-hard, even in very special cases. If, in contrast, delays occur strictly one after another and we assume that they are not known in advance, but we have to compute a new disposition timetable (based on the previous one) every time a new delay gets known, we are facing a classical online problem (see [Gat07] for an analysis of the online version of the uncapacitated delay management problem). In this case, there might exist input instances for which it is not the best approach to repeatedly compute optimal disposition timetables after the occurrence of each single source delay since other solution procedures might yield a smaller average delay. In [KS09], it is shown that in the case of uncapacitated delay management, there are some scenarios in which passenger-oriented heuristic dispatching strategies are superior to the approach of repeatedly solving the problem to optimality. However, as the limited capacity of the track system and thus the security distances between two trains and the resulting optimization potential is neglected, the results cannot be applied to the model treated in this work. Quite the contrary, our results in Section 4.3 show that the way of making the priority decisions has a large influence on the relative error and that this influence highly depends on the distribution of the source delays. It is up to now not clear which influence the priority decisions have in an online setting.

In Chapter 6, it turned out that computing a recoverable-robust timetable is a hard problem even if only a simplified nonperiodic timetabling model is considered. Many approaches on robust timetabling hence focus on special cases, for example on a single train as in [KDV07] or on heuristic approaches in the case of periodic timetabling as in [LS09]. Instead of theoretically analyzing robust timetabling, one could also take the approach to numerically compare the delay resistance of different timetables under simulated source delays and afterwards use that timetable that behaves best in average. Hence, in the next section, we present a programming framework that allows to numerically analyze the impact of different planning stages in public transportation on subsequent stages and on the operational phase. It is a joint research project that has been described in [SS09]. Parts of it are based on the results presented in this thesis.

7.3 Simulation and Numerical Evaluation with LinTim

To numerically analyze for example how delay resistant a timetable is, one could simulate different source delays and apply an optimal delay management strategy (or use the dispatching strategy that is intended to be used in practice) in each scenario. By doing so, the robustness of different timetables for common delay scenarios can be compared.

This kind of studies can be done quite comfortably using LinTim, a framework for planning in public transportation. In the following, we give a short overview of LinTim, especially of the parts which are related to this work. For a more detailled introduction, we refer to [SS09].

Overview of LinTim

The classical strategic planning steps in public transportation are network design, line planning, timetabling, vehicle scheduling, and crew scheduling. In the operational phase, delay management and crew re-scheduling are important tasks. Each of these planning steps and tasks on its own is well studied. However, the interaction between them is not understood so far.

Currently, integration of different planning steps in public transportation is an important issue. Solving integrated problems is certainly harder than solving the single optimization problems of each planning phase separately. So the question arises if it is worth to deal with the large effort of integrating the problems or if it is sufficient to solve the problems separately step by step.

To answer this question theoretically is as hard as solving the integrated problems. It hence would be nice to be able to evaluate for example timetables not only regarding the classical cost objective, but also to evaluate the performance of different timetables when setting the vehicle schedules or with respect to delay management policies in the operational phase. LinTim is a software library which is capable to perform such evaluations.

LinTim is on the one hand a collection of different algorithms for single planning steps. On the other hand, it provides tools that use the output of a preceding planning phase to create the input of the subsequent planning phase. This allows to evaluate different combinations of algorithms and to answer questions like

- From which line plans can we generate a good timetable?
- What is the impact of the lines on the vehicle schedules?
- How delay-resistant are different line concepts?
- Which timetables are suitable for generating good vehicle schedules?
- How delay-resistant are different timetables?

Structure of LinTim

We shortly sketch the workflow of LinTim. The starting point is a public transportation network (PTN) and information about the costs and the customers' demand in this network. This input data is then used by line planning procedures. They output a line concept, i.e. a set of lines together with their frequencies. Currently, six different algorithms for line planning are integrated in LinTim. From this output, LinTim creates the input for the next planning step which is to find a timetable. The input required for timetabling is the periodic event-activity network with customers' data on the activities. With this input, algorithms for timetabling can be run ending up with a timetable. The timetable allows to test different delay management policies.

In the design of the library, the main goal is that its usage and extension should be as simple as possible. This includes a modular design in which each planning step can be done separately and in which new algorithms and even new planning steps can be integrated easily. Furthermore, there are no regulations on programming languages since the communication between different steps is done through standardized plain-text data files. Running different parts of LinTim is controlled by shell scripts and makefiles or by a graphical user interface (see Figure 7.1).

Figure 7.1: Screenshot of the LinTim GUI.

Test cases

A data set is given by a PTN consisting of stations and direct connections between them. For each edge, we also need upper and lower bounds on the traveling time. Furthermore, we require an origin-destination matrix, reflecting how many passengers travel between each pair of stations, and passenger weights on the edges (as some line planning algorithms require an origin-destination matrix, while other line planning algorithm can only use traffic loads on the edges as input). Finally, each dataset has to provide maximal and minimal frequencies for each edge where the minimal frequencies reflect the minimal required number of trains on an edge to serve the customers' demand, while the maximal frequencies are needed to model capacity issues.

The test data which is available at the moment is the following:

- a hand-made example including 8 stations, 8 edges, 23 OD-pairs,
- a small real-world example with 250 stations, 326 edges, 31 000 OD-pairs, and
- a medium-size real-world example with 319 stations, 452 edges, 51 000 OD-pairs.

Evaluating a timetable's robustness with LinTim

Given a (periodic) timetable, evaluating its robustness with LinTim works as follows: As our delay management model requires an aperiodic event-activity network, the periodic event-activity network used for timetabling has to be transformed to an aperiodic one (it has to be "rolled out"). To this end, LinTim allows to define two points in time that describe the time interval that should be taken into account when rolling-out (for example all events that take place between 8 a.m. and 3 p.m.). Once these two points in time are given, rolling-out is straightforward: all (periodic) events and all (periodic) driving, waiting, and changing activities are duplicated according to their frequency and the length of the observation period. However, if two periodic events i and j are connected by a pair of periodic headway activities, then rolling-out is a little bit more complicated: each "rolled-out" copy of i and each "rolled-out" copy of j have to be connected by a pair of aperiodic headway activities, resulting in a multiplication of the number of headway activities, compared to other activities. A similar technique has been described in [LSS+10].

To evaluate the robustness, a test scenario providing source delays on events and/or on activities is needed. One possibility is to take real-world source delays if those are known for the input data of LinTim (for example if a real-world timetable and a set of delays that occurred during the operation of this timetable are given and the timetable should be compared with an alternative one, based on the same event-activity network).

Another possibility is to simulate delays. The current delay generator implemented in LinTim randomly chooses some events and/or activities and assigns a randomly chosen delay to them. The number of delays, the minimum and maximum amount of delay, and the type of delays (only on events, only on activities, or both on events and activities) can be chosen by the user as well as the period in which delays should occur (for example it is possible to take into account all events and all activities between 8 a.m. and 3 p.m. when "rolling out" the periodic timetable, but only to allow source delays to occur between 11 a.m. and 2 p.m.). Due to the modular design, other delay generators can be easily integrated, for example to simulate construction work in a station or on a track.

The solution procedures for the delay management problem that already have been implemented are:

- an exact solution procedure, solving the IP formulation (2.1)-(2.9) directly by invoking a commercial solver (FICO Xpress),
- the Fsfs, Frfs, Frfs-Fix, and Fsfs-Fix heuristics presented in Chapter 4.

Currently, we are working on integrating the remaining heuristics from Chapter 4 (No-Wait-Repair, All-Wait-Repair, and Priority-Repair) as well as the reduction techniques presented in Chapter 3.

For evaluating a timetable under delays, apart from simply giving operating figures like the objective value of the delay management stage and the number of missed connections, LinTim has a feature to visualize delays: for each station of the PTN, it sums up the delays of all events that take place in this station, then it plots the PTN and colorizes the nodes representing the stations according to the delays. An example for such a plot is given in Figure 7.2 (for the smallest test-data-set included in LinTim).

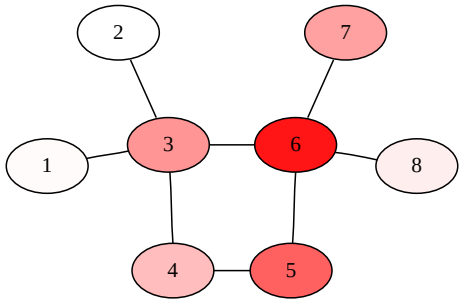

Figure 7.2: Visualizing delays with LinTim: Nodes are stations, the intensity of the color indicates the sum of all delays of all events within that station.

7.4 Possible Extensions

We conclude this work with an overview of possible extensions of our model.

7.4.1 Passenger Re-Routing

In our model, we assume that the path of each passenger is fixed, i.e. all passengers always take the lines they planned to use, even if they miss a connection and have to wait for the next train of the same line in the next period (this is why we weight the number of missed connections with a factor of T in the objective (2.1) of (DM)). However, in practice, a passenger would probably take a different connection or even change the planned path completely. If this kind of re-routing is taken into account, the weights of arrival events and changing activities no longer are fixed, but depend on which path each passenger takes – while the passengers' decisions depend on the actual delays and on the wait-depart decisions made by the controller.

In [DHSS09], a first approach for delay management with passenger re-routing is suggested. To this end, the event-activity network is extended as follows: For each OD-pair (i.e. for each pair of stations between which at least one passenger wants to travel at a given time), an *origin event* and a *destination event* are added. These events are connected to the rest of the event-activity network by two new types of activities: *origin arcs* and *destination arcs*.

However, as mentioned earlier, the uncapacitated delay management problem already is NP-hard. Adding re-routing decisions for the passengers hence further increases the complexity of delay management. With exactly one OD-pair, the problem can be solved in polynomial time; however, even the special case of more than one OD-pair where all passengers have the same origin already is NP-hard.

7.4.2 Routing of Trains in Stations

In our model, we assume that a solution of (DM) yields a feasible disposition timetable. However, our model cannot take into account all technical restrictions occurring in a railway setting. One example is the so-called *train platforming problem*: In practice, the capacity of stations (i.e. the number of platforms) is limited, and routing trains through stations and assigning trains to platforms is an independent planning step in railway optimization that is done right after computing the timetable (for a survey on the train platforming problem, see [CKM$^+$07] or [Lus08]).

When delays occur, the original assignment of trains to platforms might not be feasible any more (for example caused by a delayed train blocking the platform assigned to a punctual train), or the originally planned routing of trains through a station might get infeasible (for example due to a delayed train blocking the path of another train). Hence a disposition timetable does not only have to satisfy constraints (2.2)-(2.9), but it must permit to update the original assignment of trains to platforms and the original planned train paths to a feasible solution of the train platforming problem. Routing trains through stations is rather easy in small stations – however, as the problem is NP-hard in general (see [KRZ97]), in major stations with complex topologies, it becomes a challenging problem, and integrating it into the delay management problem further increases the complexity.

A different approach is to deal with both steps separately as it is done within the project DisKon (see [BGJ+05]): Here, in the first step, the delay management problem is solved based on a macroscopic model. Afterwards, the second step ensures feasibility (including, among many others, the constraints resulting from the train platforming problem) based on a microscopic model, using an approach similar to the relax & repair heuristics (i.e. shifting events violating some constraint into the future). However, this approach may yield rather bad solutions.

7.4.3 Maintenance Planning

In Chapter 5, we integrated rolling stock circulations into the delay management problem. However, changing the assignment of trips to trains might have an influence on maintenance planning: the length of the circulation (the sequence of all trips of one day) of an ICE train of Deutsche Bahn AG is up to 1 500 km, and each 4 000 km (i.e. each 2-3 days), a maintenance has to be carried out (see [DB06]). Thus one should take into account the maintenance plan when assigning an additional trip to a train or when making significant changes to the rolling stock circulations.

To the best of our knowledge, rolling stock circulations *and* maintenance planning have not yet been integrated into the delay management problem at the same time (in fact this is a current research project of a diploma thesis, see [Wol09]). However, there are some results on integrating rolling stock circulations and maintenance planning during the strategical planning phase, see [Bet07] and references therein.

Glossary

Notation	Description	Page
$\mathcal{A}$	activities	10
$\mathcal{A}_{\text{change}}$	changing activities	11
$\mathcal{A}_{\text{circ}}$	circulation activities	97
$\mathcal{A}_{\text{drive}}$	driving activities	11
$\mathcal{A}_{\text{head}}$	headway activities	11
$\mathcal{A}_{\text{head}}^{\text{back}}$	backward headways	14
$\mathcal{A}_{\text{head}}^{\text{forw}}$	forward headways	14
$\mathcal{A}_{\pi}$	activities respected by the timetable π	36, 102
$\mathcal{A}_{\text{reduced}}$	reduced activity set computed by algorithm Reduce	38, 102
$\mathcal{A}_{\text{wait}}$	waiting activities	11
$\mathcal{E}$	events	10
$\mathcal{E}_{\text{arr}}$	arrival events	11
$\mathcal{E}_{\text{dep}}$	departure events	11
$\mathcal{E}_{\text{end}}$	set of all last events of all trips	95
$\mathcal{E}_{\text{init}}$	virtual events modeling a train leaving some depot	97
$\mathcal{E}_{\text{mark}}$	potentially delayed events	36
$\mathcal{E}_{\text{reduced}}$	reduced event set computed by algorithm Reduce	38, 102
$\mathcal{E}_{\text{start}}$	set of all first events of all trips	95
$\mathcal{E}_{\text{term}}$	virtual events modeling a train driving back to some depot	97

Notation	Description	Page
$\mathcal{N}$	event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$	10
$\mathcal{N}_{\text{reduced}}$	reduced event-activity network computed by algorithm REDUCE	38, 102
$\text{pre}(v, E')$	predecessors of v in G'	10
$\text{suc}(v, E')$	successors of v in G'	10

List of Algorithms

List of Optimization Problems

Problem	Description	Page
BDM	bounded delay management problem	32
BDMC	delay management problem with integrated rolling stock circulations and bounded delay	101
DM	capacitated delay management problem	17
DMC	delay management problem with integrated rolling stock circulations	98
PP	project planning problem	20
Re-Sched	capacitated re-scheduling problem	19
TT	timetabling problem	122
TT^a	timetabling problem with arc weights	122
$\mathcal{TT}^a_1$	robust timetabling with arc weights and limited-delay	125
$\mathcal{TT}^a_\sigma$	multi-stage recoverable-robust timetabling with arc weights and limited-delay	139
TT^v	timetabling problem with nonnegative node weights	122
$\mathcal{TT}^v_1$	robust timetabling with nonnegative node weights and limited-events	124
$\mathcal{TT}^v_\sigma$	multi-stage recoverable-robust timetabling with non-negative node weights and limited-events	139
UDM	uncapacitated ("pure") delay management problem	19

Bibliography

[Ass80] Arjang A. Assad. Models for rail transportation. *Transportation Research Part A*, 14(3):205–220, June 1980.

[BDM09] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009.

[BEN06] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski, editors. *Robust Optimization*, volume 107 of *Mathematical Programming*. Springer, 2006.

[Bet07] Johanna Betz. *Mathematische Modelle zur Umlauf-Wartungs-Planung im Schienenverkehr und deren Lösung*. Diploma thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2007. In German.

[BGJ+05] Nicolai Bissantz, Sabine Güttler, Jürgen Jacobs, Stephan Kurby, Thorsten Schaer, Anita Schöbel, and Susanne Scholl. DisKon – Laborversion eines flexiblen, modularen und automatischen Dispositionsassistenzsystems. *Eisenbahntechnische Rundschau (ETR)*, 45(12):809–821, 2005. In German.

[BHK02] Peter Brucker, Silvia Heitmann, and Sigrid Knust. Scheduling railway traffic at a construction site. *OR Spectrum*, 24(1):19–30, 2002.

[BHLS08] Andre Berger, Ralf Hoffmann, Ulf Lorenz, and Sebastian Stiller. TOPSU – RDM A simulation platform for online railway delay management. In *SimuTools – Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. ICST, 2008.

[BL97] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer, 1997.

[BS04] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[BS07] Hans-Georg Bayer and Bernhard Sendhoff. Robust optimization – A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, July 2007.

[CBH05] Gabrio Caimi, Dan Burkolter, and Thomas Herrmann. Finding delay-tolerant train routings in stations. In *Operations Research Proceedings 2004*, pages 136–143. Springer, 2005.

[CDD+07] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Robust algorithms and price of robustness in shunting problems. In *ATMOS 2007 – 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Research Online Publication Server (DROPS), 2007.

[CDD+08] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Delay management problem: Complexity results and robust algorithms. In *Proceedings of the 2nd Annual International Conference on Combinatorial Optimization and Applications (COCOA'08)*, volume 5165 of *Lecture Notes in Computer Science*, pages 458–468, 2008.

[CDD+09a] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229–257, October 2009.

[CDD+09b] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, Alfredo Navarra, Michael Schachtebeck, and Anita Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 28–60. Springer, 2009.

[CDSS08] Serafino Cicerone, Gabriele Di Stefano, Michael Schachtebeck, and Anita Schöbel. Dynamic algorithms for recoverable robustness problems. In *ATMOS 2008 – 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Research Online Publication Server (DROPS), 2008.

[CFB+09] Gabrio Caimi, Martin Fuchsberger, Dan Burkolter, Thomas Herrmann, Raimond Wüst, and Samuel Roos. Conflict-free train scheduling in a compensation zone exploiting the speed profile. In *Proceesings of ISROR*, February 2009.

[CG94] Xiaoqiang Cai and C. J. Goh. A fast heuristic for the train scheduling problem. *Computers & Operations Research*, 21(5):499–510, May 1994.

[CGD09] Francesco Corman, Rob M. P. Goverde, and Andrea D'Ariano. Rescheduling dense train traffic over complex station interlocking areas. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 369–386. Springer, 2009.

[CGM98] Xiaoqiang Cai, C. J. Goh, and Alistair I. Mees. Greedy heuristics for rapid scheduling of trains on a single track. *IIE Transactions*, 30(5):481–493, May 1998.

[CKM+07] Alberto Caprara, Leo G. Kroon, Michele Monaci, Marc Peeters, and Paolo Toth. Passenger railway optimization. In *Handbooks in Operations Research and Management Science: Transportation*, volume 14, pages 129–187. Elsevier, 2007.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[Con08] Carla Conte. *Identifying dependencies among delays.* PhD thesis, Georg-August-Universität Göttingen, 2008.

[CS07] Carla Conte and Anita Schöbel. Identifying dependencies among delays. In *proceedings of IAROR 2007*, 2007.

[CTV98] Jean-François Cordeau, Paolo Toth, and Daniele Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, November 1998.

[DB06] Deutsche Bahn AG. Der Innovationstreiber. *bahntech – Das Technik-Magazin der Deutschen Bahn AG*, 01/06:4–13, 2006. In German.

[DCPP08] Andrea D'Ariano, Francesco Corman, Dario Pacciarelli, and Marco Pranzo. Reordering and local rerouting strategies to manage train traffic in real time. *Transportation Science*, 42(4):405–419, November 2008.

[DDD98] Remco De Vries, Bart De Schutter, and Bart De Moor. On max-algebraic models for transportation networks. In *Proceedings of the International Workshop on Discrete Event Systems*, pages 457–462, 1998.

[DDNP09] Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, and Cristina M. Pinotti. Recoverable robust timetables on trees. In *Third International Conference on Combinatorial Optimization and Applications (COCOA 2009)*, volume 5573 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2009.

[DHL08] Luigi De Giovanni, Géraldine Heilporn, and Martine Labbé. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, September 2008.

[DHSS09] Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay management wit re-routing of passengers. In *ATMOS 2009 – 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Research Online Publication Server (DROPS), 2009.

[DLZL06] Maged M. Dessouky, Quan Lu, Jiamin Zhao, and Robert C. Leachman. An exact solution procedure for determining the optimal dispatching times for complex rail networks. *IIE Transactions*, 38(2):141–152, 2006.

[DP04] Andrea D'Ariano and Marco Pranzo. A real time train dispatching system based on blocking time theory. In *Proceedings of the 8th TRAIL Annual Congress*, pages 129–152, 2004.

[DPP07] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, December 2007.

[EK04] Ophelia Engelhardt-Funke and Michael Kolonko. Analysing stability and investments in railway networks using advanced evolutionary algorithms. *International Transactions in Operational Research*, 11(4):381–394, July 2004.

[Elm77] Salah E. Elmaghraby. *Activity Networks.* Wiley, 1977.

[FGGN09] Holger Flier, Rati Gelashvili, Thomas Graffagnino, and Marc Nunkesser. Mining railway delay dependencies in large-scale real-world delay data. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2009.

[FM06] Matteo Fischetti and Michele Monaci. Robust optimization through branch-and-price. In *Proceedings of the 37th Annual Conference of the Italian Operations Research Society (AIRO)*, 2006.

[FM09] Matteo Fischetti and Michele Monaci. Light robustness. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 61–84. Springer, 2009.

[FNSS07] Holger Flier, Marc Nunkesser, Michael Schachtebeck, and Anita Schöbel. Integrating rolling stock circulation in the delay management problem. NAM preprint series, Georg-August-Universität Göttingen, 2007.

[FSZ09] Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. Fast approaches to improve the robustness of a railway timetable. *Transportation Science*, 43(3):321–335, August 2009.

[Gat07] Michael Gatto. *On the impact of uncertainty on some optimization problems: Combinatorial aspects of delay management and robust online scheduling.* PhD thesis, ETH Zürich, 2007.

[GGJ+04] Michael Gatto, Björn Glaus, Riko Jacob, Leon Peeters, and Peter Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Proc. 9th Scand. Workshop on Algorithm Theory (SWAT)*, volume 3111 of *Lecture Notes in Computer Science*, pages 199–211, 2004.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[GJPS05] Michael Gatto, Riko Jacob, Leon Peeters, and Anita Schöbel. The computational complexity of delay management. In *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, 2005.

[GJPW07] Michael Gatto, Riko Jacob, Leon Peeters, and Peter Widmayer. Online delay management on a single train line. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 306–320. Springer, 2007.

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

[Gov05] Rob M.P. Goverde. *Punctuality of railway operations and timetable stability analysis.* PhD thesis, Delft University of Technology, 2005.

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

[Güt06] Sabine Güttler. *Statistical Modeling of Railway Data*. Diploma thesis, Georg-August-Universität Göttingen, 2006.

[HdV01] Bernd Heidergott and Remco de Vries. Towards a (max,+) control theory for public transportation networks. *Discrete Event Dynamic Systems*, 11(4):371–398, October 2001.

[Her06] Thomas Herrmann. *Stability of Timetables and Train Routings through Station Regions*. PhD thesis, ETH Zurich, 2006.

[HKF96] Andrew Higgins, Erhan Kozan, and Luis Ferreira. Optimal scheduling of trains on a single line track. *Transportation Research Part B*, 30(2):147–161, April 1996.

[HOW06] Bernd Heidergott, Geert Jan Olsder, and Jacob van der Woude. *Max Plus at Work*. Princeton University Press, 2006.

[Jac03] Jürgen Jacobs. *Rechnergestützte Konfliktermittlung und Entscheidungsunterstützung bei der Disposition des Zuglaufes*. PhD thesis, RWTH Aachen, 2003. In German.

[Jac08] Jürgen Jacobs. Rescheduling. In *Railway Timetable & Traffic*, pages 182–191. Eurailpress, 2008.

[Job08] Christoph Jobmann. *Ganzzahlige Programmierung und kernbasierte Lernverfahren für das Anschlusssicherungsproblem*. Diploma thesis, Georg-August-Universität Göttingen, 2008. In German.

[KDV07] Leo G. Kroon, Rommert Dekker, and Michiel J.C.M. Vromans. Cyclic railway timetabling: a stochastic optimization approach. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 41–66. Springer, 2007.

[KHM08] Leo G. Kroon, Dennis Huisman, and Gábor Maróti. Optimisation models for railway timetabling. In *Railway Timetable & Traffic*, pages 135–154. Eurailpress, 2008.

[KKM94] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(2):275–283, 1994.

[Kli00] Natalia Kliewer. *Mathematische Optimierung zur Unterstützung kundenorientierter Disposition im Schienenverkehr*. Diploma thesis, Universität Paderborn, 2000. In German.

[KMH+08] Leo G. Kroon, Gábor Maróti, Mathijn Retel Helmrich, Michiel Vromans, and Rommert Dekker. Stochastic improvement of cyclic railway timetables. *Transportation Research Part B*, 42(6):553–570, July 2008.

[Koc00] Wilfried Koch. A low cost tool for rescheduling in regional public transport systems. In *Proceedings of the 9th IFAC Symposium on Control in Transport Systems*, 2000.

[KRZ97] Leo G. Kroon, H. Edwin Romeijn, and Peter J. Zwaneveld. Routing trains through railway networks: Complexity issues. *European Journal of Operational Research*, 98(3):485–498, May 1997.

[KS09] Natalia Kliewer and Leena Suhl. A note on the online nature of the railway delay management problem. *Networks*, 2009. To appear.

[Kuh55] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[KW94] Peter Kall and Stein W. Wallace. *Stochastic Programming*. Wiley, 1994.

[Lie06] Christian Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technische Universität Berlin, 2006.

[LJN00] Ladislav Lettovský, Ellis L. Johnson, and George L. Nemhauser. Airline crew recovery. *Transportation Science*, 34(4):337–348, November 2000.

[LLMS09] Christian Liebchen, Marco Lübbecke, Rolf H. Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2009.

[LS09] Christian Liebchen and Sebastian Stiller. Delay resistant timetabling. *Public Transport*, 1(1):55–72, May 2009.

[LSS+10] Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and Andre Prigge. Computing delay resistant railway timetables. *Computers & Operations Research*, 37(5):857–868, May 2010. To appear.

[LTW63] Ferdinand K. Levy, Gerald L. Thompson, and Jerome D. Wiest. ABCs of the critical path method. *Harvard Business Review*, 41(5):98–108, October 1963.

[Lus08] Richard M. Lusby. *Optimization methods for routing trains through railway junctions*. PhD thesis, University of Auckland, 2008.

[Mat05] Lars-Göran Mattsson. Ways of deriving train delay relationships. In *Proceedings of the 1st International Seminar on Railway Operations Modelling and Analysis*, 2005.

[MP02] Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, December 2002.

[MPP04] Alessandro Mascis, Dario Pacciarelli, and Marco Pranzo. Scheduling models for short-term railway traffic optimization. In *Proceedings of the 9th International Conference on Computer-Aided Scheduling of Public Transport*, 2004.

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

[NH04] Nils O.E. Olsson and Hans Haugland. Influencing factors on train punctuality – Results from some Norwegian studies. *Transport Policy*, 11(4):387–397, October 2004.

[NO00] David Nelson and Kay O'Neil. Commuter rail service reliability: On-time performance and causes for delays. *Transportation Research Record: Journal of the Transportation Research Board*, 1704:42–50, 2000.

[NYN+05] Tomii Norio, Tashiro Yoshiaki, Tanabe Noriyuki, Hirai Chikara, and Muraki Kunimitsu. Train rescheduling algorithm which minimizes passengers' dissatisfaction. In *18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 3533 of *Lecture Notes in Computer Science*, pages 829–838, 2005.

[Nys05] Birre Nyström. *Punctuality and railway maintenance.* Licentiate thesis, Luleå University of Technology, 2005.

[Nys08] Birre Nyström. *Aspects of improving punctuality.* PhD thesis, Luleå University of Technology, 2008.

[PALF01] Li Ping, Nie Axin, Jia Limin, and Wang Fuzhang. Study on intelligent train dispatching. In *Proceedings of IEEE Intelligent Transportation Systems Conference*, 2001.

[Roc84] R. Tyrrell Rockafellar. *Network flows and monotropic optimization.* Wiley, 1984.

[RS64] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical report, SEMA, Note D.S., No. 9, Paris, 1964. In French.

[RS03] Andrzej Ruszczynski and Alexander Shapiro, editors. *Stochastic Programming*, volume 10 of *Handbooks in Operations Research and Management Science.* Elsevier, 2003.

[SBK01] Leena Suhl, Claus Biederbick, and Natalia Kliewer. Design of customer-oriented dispatching support for railways. In *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economics and Mathematical Systems*, pages 365–386. Springer, 2001.

[Sch01a] Anita Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

[Sch01b] Susanne Scholl. *Anschlusssicherung bei Verspätungen im ÖPNV.* Diploma thesis, Universität Kaiserslautern, 2001. In German.

[Sch06] Anita Schöbel. *Optimization in public transportation.* Optimization and Its Applications. Springer, 2006.

[Sch07] Anita Schöbel. Integer programming approaches for solving the delay management problem. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 145–170. Springer, 2007.

[Sch09a] Michael Schachtebeck. Algorithmic approaches to the capacitated delay management problem. In *CASPT 09 – 11th Conference on Advanced Systems for Public Transport*, July 2009.

[Sch09b] Anita Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, June 2009.

[SK06] Sergey Shebalov and Diego Klabjan. Robust airline crew pairing: Move-up crews. *Transportation Science*, 40(3):300–312, August 2006.

[SK09] Anita Schöbel and Albrecht Kratz. A bicriteria approach for robust timetabling. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 119–144. Springer, 2009.

[SM97] Leena Suhl and Taïeb Mellouli. Supporting planning and operation time control in transportation systems. In *Selected Papers of the Symposium on Operations Research (SOR 96)*, Operations Research Proceedings 1996, pages 374–379. Springer, 1997.

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

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

[SS06] Anita Schöbel and Silvia Schwarze. A game-theoretic approach to line planning. In *ATMOS 2006 – 6th Workshop on Algorithmic Methods and Models for Optimization of Railways*, Dagstuhl Research Online Publication Server (DROPS), 2006.

[SS08] Michael Schachtebeck and Anita Schöbel. IP-based techniques for delay management with priority decisions. In *ATMOS 2008 – 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Research Online Publication Server (DROPS), 2008.

[SS09] Michael Schachtebeck and Anita Schöbel. Lintim – A toolbox for the experimental evaluation of the interaction of different planning stages in public transportation. Technical Report ARRIVAL-TR-0206, ARRIVAL Project, 2009.

[SS10] Michael Schachtebeck and Anita Schöbel. To wait or not to wait and who goes first? Delay management with priority decisions. Accepted for publication in *Transportation Science*, 2010.

[SU89] Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

[SW83] Richard L. Sauder and William M. Westerman. Computer aided train dispatching: Decision support through optimization. *Interfaces*, 13(6):24–37, December 1983.

[SW04] Eckehard Schnieder and Stefan Wegele. Dispatching of train operations using genetic algorithms. In *Proceedings of the 9th International Conference on Computer-Aided Scheduling of Public Transpor*, 2004.

[Szp73] Bernardo Szpigel. Optimal train scheduling on a single track railway. In *Operational Research '72: Proceedings of the 6th IFORS International Conference on Operational Research*, pages 343–352. North-Holland Publishing Company, 1973.

[Tör06] Johanna Törnquist. Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *ATMOS 2005 – 5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, Dagstuhl Research Online Publication Server (DROPS), 2006.

[Tör07] Johanna Törnquist. Railway traffic disturbance management – An experimental analysis of disturbance complexity, management objectives and limitations in planning horizon. *Transportation Research Part A*, 41(3):249–266, March 2007.

[Tör08] Johanna Törnquist-Krasemann. Dynamic railway traffic management during disturbances: Focus on the complexity imposed by deregulation. Presented at the 15th World Congress on Intelligent Transport Systems, November 2008.

[TP07] Johanna Törnquist and Jan A. Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B*, 41(3):342–362, March 2007.

[Wol09] Julian Wolff. *Integrating Delay Management, Rolling Stock Circulations, and Maintenance Planning.* Diploma thesis, Georg-August-Universität Göttingen, 2009. To be submitted.

[Yua06] Jianxin Yuan. *Stochastic modeling of train delays and delay propagation in stations.* PhD thesis, Delft University of Technology, 2006.

[Zas00] Kai Frederik Zastrow. *Analyse und Simulation von Entstörungsstrategien bei der Automatisierung von U-Bahnsystemen.* PhD thesis, TU Berlin, 2000. In German.

Curriculum vitae – Lebenslauf

Persönliche Daten

Name	Michael Schachtebeck
Geburtsdatum	03. November 1978
Geburtsort	Northeim
Familienstand	verheiratet
Staatsangehörigkeit	deutsch
Wohnsitz	Göttingen

Akademische Ausbildung und Tätigkeiten

06/1998	Abitur am Otto-Hahn-Gymnasium Göttingen
10/1999 – 09/2001	Studium der Physik an der Georg-August-Universität Göttingen
10/2001 – 09/2004	Studium der Angewandten Informatik an der Georg-August-Universität Göttingen Abschluss: Bachelor of Science
10/2004 – 08/2006	Studium der Angewandten Informatik an der Georg-August-Universität Göttingen Abschluss: Master of Science
seit 09/2006	Wissenschaftlicher Mitarbeiter am Institut für Numerische und Angewandte Mathematik der Georg-August-Universität Göttingen
	Promotionsstudium an der Fakultät für Mathematik und Informatik an der Georg-August-Universität Göttingen