Open access • Book Chapter • DOI:10.1007/978-3-540-74247-0_7

Integer programming approaches for solving the delay management problem

Anita Schöbel

Institutions: University of Göttingen

Published on: 20 Jun 2004 - Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

Topics: Integer programming, Linear programming, Integer (computer science), Time complexity and Equivalence (measure theory)

Integer programming approaches for solving the delay management problem

Anita Schöbel

Institute for Numerical and Applied Mathematics, Georg-August University, Göttingen
schoebel@math.uni-goettingen.de

Abstract. The *delay management problem* deals with reactions in case of delays in public transportation. More specifically, the aim is to decide if connecting vehicles should wait for delayed feeder vehicles or if it is better to depart on time. As objective we consider the convenience over all customers, expressed as the average delay of a customer when arriving at his or her destination.

We present path-based and activity-based integer programming models for the delay management problem and show the equivalence of these formulations. Based on these, we present a simplification of the (cubic) activity-based model which results in an integer *linear* program. We identify cases in which this linearization is correct, namely if the so-called *never-meet property* holds. We analyze this property using real-world railway data. Finally, we show how to find an optimal solution in linear time if the never-meet property holds.

1 Introduction

A major reason for complaints about public transportation is the missing punctuality, which—unfortunately—is a fact in many transportation systems. Since it seems to be impossible to avoid delays completely, it is a necessary issue in the operative work of a public transportation company to deal with delayed vehicles. In this paper we focus on the convenience of the customers and present a model for minimizing the average delay over all passengers.

Let us consider some vehicle (e.g., a train g) that arrives at a station with a delay. At the station, there are other vehicles (e.g., buses h and h') ready to depart, see Figure 1. What should each of these connecting vehicles do? There are two alternatives:

- A connecting vehicle h can **wait** to allow passengers to change from the delayed vehicle g to h.
- The connecting vehicle h can **depart** on time.

Unfortunately, both decisions have negative effects: In the first case, vehicle h causes a delay for passengers already within h, but also for customers who wish

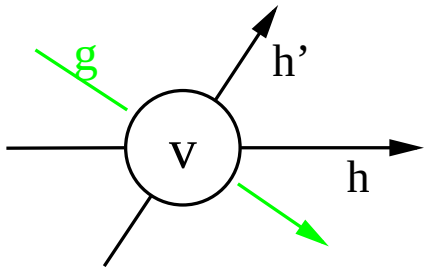

Fig. 1. The wait-depart decision at one single station.

to board vehicle h later on, and possibly for subsequent other vehicles which will have to wait for its delay. In the second case, however, all customers who planned to change from the delayed vehicle g into h will miss their connection.

In the first case the connecting vehicle h does not depart at its scheduled time, but with a delay. The new departure time of h is called its *perturbed* timetable. In the second case, the *perturbed* departure time of h at v equals the scheduled one.

In case of some known delays, the *delay management problem* is to find wait-depart decisions and a perturbed timetable for all vehicles in the network, such that the sum of all delays over all customers is minimized. The delay of a customer is defined as the delay he has when he reaches his destination. Recently the NP-completeness of this problem has been shown (see [GJPS05]).

Since in the delay management problem new departure times for each vehicle at each station have to be determined, it is related to finding timetables in public transportation. In this field, a lot of research has been done for periodic and non-periodic timetables. An excellent overview on periodic timetabling is given by [Pee03]. We also refer to [Nac98,Car99,Gov98,vE01] and references therein. However, note the main difference between timetabling and delay management: In the timetabling problem the connections are given in advance, while in the delay management problem we have not only to find a (perturbed) timetable, but also to decide which connections should be maintained and which can be dropped.

How to *react* in case of delays has due to the size and complexity of the problem been mainly tackled by simulation and expert systems. We refer to [SBK01,SMBG01] for providing a knowledge-based expert system including a simulation of wait-depart decisions with a *what-if* analysis. Simulation has also been used in [Ack99,SM01].

In [GS02] the delay management problem has been formulated as a bicriteria problem, minimizing the number of missed connections and the delay of the vehicles simultaneously, and solved by methods of project planning. The weighted sum of these functions has been minimized in [RdVM98] by an enumeration procedure and a by greedy heuristic within a max-plus algebraic model, see

also [SvdB01]. Dynamic programming has been used in [GGJ+04] to identify polynomially solvable cases.

Integer programming formulations so far only exist as first attempts for the simple case without slack times, assuming that the customers on each edge are fixed (see the diploma theses of [Kli00] and [Sch01a]). In Section 4 we are able to identify cases in which such models are correct. A first *exact* linear integer model for the delay management problem is presented in [Sch01b], and will be reviewed in this paper in a more convenient notation at the beginning of Section 3. A detailed description of the delay management problem will be published in [Sch06]. Based on the formulation (TDM-B) presented in this text, [GHL05] developed two new formulations reducing the number of variables in the models.

Related also work includes how to reduce delays by investing into new tracks ([EFK01,EF02]), how to minimize the sum of waiting times of customers at their starting stations in a stochastic context ([APW02]), and a first on-line model of the problem along a line ([GJPW04]).

The aim of this paper is to present a new and more general integer programming formulation of the delay management problem, for which we are still able to develop solution approaches. Although our model can be applied to many different objective functions we specialize here on minimizing the sum of all delays over all customers. After introducing definitions and basic properties in Section 2 we develop a new integer programming formulation for the delay management problem in Section 3. In Section 4 we show that this formulation can be linearized if a special condition, called *the never-meet property* holds. We analyze this property in Section 5 using real-world data of the largest German railway company, *Deutsche Bahn*. In Section 6 we show how to solve the delay management problem in linear time in this case. The paper is concluded by some remarks on future research.

2 Notation, concepts, and basic properties

We first introduce a new notation for the delay management problem, based on its representation as an *activity-on-arc project network* (see e.g. [Nac98] for using this concept in timetabling).

We denote an arrival of a vehicle g at a station v as *arrival event* (g, v, arr), while a *departure event* (g, v, dep) describes the departure of some vehicle g at some station v. The *event activity network* is a graph $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ where

- $\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$ is the set of all arrival and all departure events

– $\mathcal{A} = \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \cup \mathcal{A}_{change}$ is a set of directed arcs, called *activities*, defined by

$$\begin{aligned}
\mathcal{A}_{wait} &= \{((g, v, \text{arr}), (g, v, \text{dep})) \in \mathcal{E}_{arr} \times \mathcal{E}_{dep}\} \\
\mathcal{A}_{drive} &= \{((g, v, \text{dep}), (g, u, \text{arr})) \in \mathcal{E}_{dep} \times \mathcal{E}_{arr} : \text{ vehicle } g \text{ goes} \\
&\qquad \text{directly from station } v \text{ to } u\}, \\
\mathcal{A}_{change} &= \{((g, v, \text{arr}), (h, v, \text{dep})) \in \mathcal{E}_{arr} \times \mathcal{E}_{dep} : \text{ a changing} \\
&\qquad \text{possibility from vehicle } g \text{ into } h \text{ at station } v \text{ is required}\}.
\end{aligned}$$

The driving and waiting activities are performed by vehicles, while the changing activities are used by the customers. As an example, a small event-activity network is depicted in Figure 2.

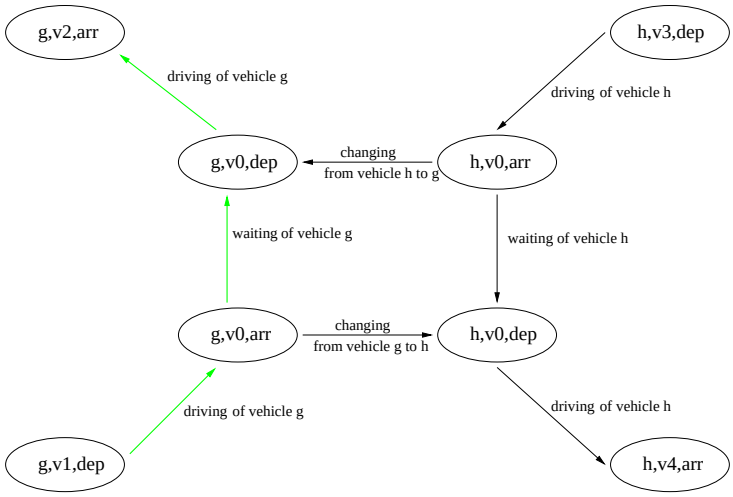

Fig. 2. An event-activity network.

Note that $\mathcal{N}$ is a special case of a time-expanded network and hence contains no directed cycles. This means that a precedence relation $\prec$ between events (or activities) is canonically given, where $i \prec j$ hence indicates that there exists a path from i to j. We remark that for a given set of events (or of activities) a minimal element w.r.t. $\prec$ always exists, but it needs not be unique.

Using the notation of event-activity networks, a *timetable* $\Pi \in \mathbb{Z}^{|\mathcal{E}|}$ is given by assigning a time Π_i to each event $i \in \mathcal{E}$ (see [Nac98]). Timetables are usually given in minutes and hence consist of integer values. The planned duration of activity $a = (i, j)$ is given by $\Pi_j - \Pi_i$. Furthermore, let $L_a \in \mathbb{N}$ be the minimal duration needed for performing activity a. We assume that the timetable is *feasible*, i.e.,

$$\Pi_j - \Pi_i \geq L_a \text{ for all } a = (i, j) \in \mathcal{A}.$$

We further assume that *source delays* are known at some of the events, where they might have occurred at the preceding activity or at the event itself. Let $\mathcal{SD} \subseteq \mathcal{E}_{arr}$ denote the set of source-delayed events, and $d_i > 0$ indicate the delay they have. (For $i \notin \mathcal{SD}$ the source delay $d_i = 0$.)

If source delays occur, some of the subsequent arrival and departure times Π_i can also not take place punctually, since the minimal durations L_a for subsequent activities have to be taken into account. The outcome $\Pi + y$ is called a *perturbed timetable*, and y_i is called the delay of event i. Such a perturbed timetable is feasible, if

- the source delays are taken into account, i.e., $\Pi_i + y_i \geq \Pi_i + d_i$, and
- the delay is carried over correctly from one event to the next, i.e., $\Pi_j + y_j - (\Pi_i + y_i) \geq L_a$ holds for all driving and waiting activities $a = (i, j)$.

Defining the *slack time* s_a of an activity $a \in \mathcal{A}$ as the time which can be saved when performing activity a as fast as possible, i.e.,

$$s_a = \Pi_i - \Pi_j - L_a$$

we can equivalently restate the two above conditions in terms of the delay vector y as follows.

Definition 1. *A set of delays y_i for all $i \in \mathcal{E}$ is* **feasible***, if*

$$y_i \geq d_i \text{ for all } i \in \mathcal{E} \quad \text{and} \tag{1}$$

$$y_i - y_j \leq s_a \quad \text{for all } a = (i, j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}. \tag{2}$$

Condition (2) makes sure that the delay at the start of activity a is transferred to its end, where it can be reduced by the slack time of a.

In the following we only use the slack times s and the delays y instead of the minimal durations L and the timetable Π.

Definition 1 only takes the driving and waiting activities into account. However, in the delay management problem the goal is to identify which changing activities should be maintained and which ones can be dropped. For a changing activity we analogously require that

$$y_i - y_j \leq s_a \quad \text{if } a = (i, j) \text{ is maintained} \tag{3}$$

We are now in the position to specify feasible solutions of the delay management problem.

Definition 2. *A set of connections $\mathcal{A}^{fix} \subseteq \mathcal{A}_{change}$ together with a feasible set of delays y_i for all $i \in \mathcal{E}$ is a* **feasible solution of the delay management problem***, if*

$$y_i - y_j \leq s_a \quad \text{for all } a = (i, j) \in \mathcal{A}^{fix},$$

i.e., for all connections $a \in \mathcal{A}^{fix}$ which are maintained.

Note that a timetable would also be feasible if some vehicles depart or arrive late without any reason. Such solutions are clearly not optimal. The "most punctual" solutions are defined below.

Definition 3. *Let $(\mathcal{A}^{fix}, y)$ be a feasible solution of the delay management problem. The delay y is called* **time-minimal** *with respect to $\mathcal{A}^{fix}$ if all feasible solutions $(\mathcal{A}^{fix}, y')$ satisfy $y \leq y'$ (where as usual $\leq$ is meant component-wise). We write $y(\mathcal{A}^{fix})$.*

A time minimal solution with respect to each set $\mathcal{A}^{fix} \subseteq \mathcal{A}_{change}$ can be found efficiently by using the critical path method (CPM) of project planning.

To this end, we transform the event-activity network into a project network (as defined, e.g., in [Elm77]) by introducing one super-source s and taking

$$\mathcal{A}(\mathcal{A}^{fix}) = \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \cup \mathcal{A}^{fix}$$

and additional *timetable activities* $\{(s,i) : i \in \mathcal{E}\}$ as set of activities in the corresponding project network. The duration of an activity is set to L_a for $a \in \mathcal{A}$ and to the scheduled timetable Π_i if $a = (s,i)$. Then the earliest possible starting time of each activity is a time-minimal solution of the delay management problem. The following procedure uses the critical path method to determine the earliest starting times but is applied directly in the notation of slack times s and delays y according to Definition 1.

Algorithm 1: Calculating a time-minimal solution for a set $\mathcal{A}^{fix}$

Input: $\mathcal{N}$, d_i, s_a, $\mathcal{A}^{fix}$.
Output: `Optimal (time-minimal) solution w.r.t.` $\mathcal{A}^{fix}$.
Step 1. `Sort` $\mathcal{E} = \{i_1, \ldots, i_{|\mathcal{E}|}\}$ `according to` $\prec$.
Step 2. `For` $k = 1, \ldots, |\mathcal{E}|$: $y_{i_k} = \max\{d_{i_k}, \max_{a=(i,i_k)\in\mathcal{A}(\mathcal{A}^{fix})} y_i - s_a\}$
Step 3. `Output:` $y_i,\ i \in \mathcal{E}$

By induction it is easy to show that the time-minimal solution $y(\mathcal{A}^{fix})$ with respect to each set $\mathcal{A}^{fix} \subseteq \mathcal{A}_{change}$ is unique, and that it has the following two properties:

1. $\mathcal{A}^1 \subseteq \mathcal{A}^2 \subseteq \mathcal{A}_{change} \implies y(\mathcal{A}^1) \leq y(\mathcal{A}^2)$. i.e. the delays get smaller if connections are dropped, and
2. $y = y(\mathcal{A}^{fix})$ satisfies $y_i \leq D = \max\{d_i : i \in \mathcal{E}\}$ for all $i \in \mathcal{E}$, i.e. the maximal delay of a single event in a time-minimal solution is bounded by the largest given source delay.

Other approaches for calculating time minimal solution sand the details of the proofs can be found in [Sch06].

As mentioned before, our objective is to minimize the sum of all delays over all customers. To this end, we first specify the customers' data.

A customer's paths is given as a sequence of events, i.e.,

$$p = (i_1, i_2, \ldots, i_{p_L})$$

where $i_k \in \mathcal{E}$ are events, and $(i_k, i_{k+1}) \in \mathcal{A}$ are activities. We will write $a = (i_k, i_{k+1}) \in p$ in this case. Note that i_1 is a departure event, i_2 an arrival event, $i_3 \in \mathcal{E}_{dep}$ and so on. Furthermore, $i(p)$ denotes the last event on path p and w_p the number of passengers who want to use path p. We denote $\mathcal{P}$ as the set of all customers' paths.

To calculate the delay of a passenger on path p we need the following two basic assumptions:

1. There is one (common) time period T for all vehicles.
2. In the next time period all vehicles are on time.

In praxis, both assumptions are usually not satisfied. The first of them can be relaxed a bit, allowing different periods for each of the activities. Taking the largest of the periods of all lines overestimates the delay, but seems to be a reasonable approach. The second assumption is accepted by practitioners since the planning period in on-line disposition is usually less than the time period T (often one hour). It is an open problem to deal with future delays by using stochastic optimization.

To calculate the delay of a customer using some path $p \in \mathcal{P}$, we have to distinguish the following two cases.

Case 1: If all connections of path p are maintained (i.e., the path is *maintained*), the delay of a passenger on path p is the arrival delay $y_{i(p)}$ of his last event $i(p)$.

Case 2: If at least one connection of path p is missed, the delay of a passenger on path p is given by T.

We are finally in the position to define the **(total) delay management problem**.

(TDM): *Given $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, slack times s_a for all $a \in \mathcal{A}$, source delays $d_i, i \in \mathcal{E}$ and a set of weighted paths $\mathcal{P}$, find a feasible pair $\mathcal{A}^{fix} \subseteq \mathcal{A}_{change}$ with delays y_i, $i \in \mathcal{E}$ such that the sum of all delays over all customers is minimal.*

3 Models for delay management

As first model we present a **path-oriented** description of (TDM) (based on the formulation in [Sch01b]) which uses the following variables

$$z_p = \begin{cases} 0 & \text{if all connections on path } p \text{ are maintained} \\ 1 & \text{otherwise} \end{cases}$$

(TDM-A)

$$\min f_{\text{TDM-A}} = \sum_{p \in \mathcal{P}} w_p (y_{i(p)}(1 - z_p) + T z_p)$$

such that

$$y_i \geq d_i \quad \text{for all } i \in \mathcal{SD} \tag{4}$$
$$y_i - y_j \leq s_a \quad \text{for all } a = (i,j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \tag{5}$$
$$-Mz_p + y_i - y_j \leq s_a \quad \text{for all } p \in \mathcal{P}, a = (i,j) \in p \cap \mathcal{A}_{change} \tag{6}$$
$$y_i \in \mathbb{N} \quad \text{for all } i \in \mathcal{E} \tag{7}$$
$$z_p \in \{0,1\} \quad \text{for all } p \in \mathcal{P}, \tag{8}$$

where $M \geq D = \max\{d_i : i \in \mathcal{E}\}$.

The first two constraints (4) and (5) are the same as (1) and (2). Constraint (6) makes sure that all connections on a maintained path (i.e. a path with $z_a = 0$) satisfy (3). Finally, the objective function sums up the delay according to the two cases mentioned on page 7.

As already shown in [Sch01b], this formulation of model (TDM-A) can be linearized by substituting the quadratic terms $y_{i(p)}(1-z_p)$ by additional variables q_p, leading to the following model.

(TDM-B)

$$\min f_{\text{TDM-B}} = \sum_{p \in \mathcal{P}} w_p(q_p + Tz_p)$$

such that (4) – (8) hold, and such that

$$-Mz_p + y_{i(p)} - q_p \leq 0 \quad \text{for all } p \in \mathcal{P} \tag{9}$$
$$q_p \geq 0 \quad \text{for all } p \in \mathcal{P} \tag{10}$$

The linear formulation is significantly weaker than the quadratic formulation (TDM-A), due to the fact that the feasible set of the linear programming relaxation increased. More intuitively, one would like to use variables $\bar{z}_a$ determining if a connection $a \in \mathcal{A}_{change}$ should be maintained or not. This yields a stronger activity-based formulation for (TDM) which is derived next.

In the **activity-based** model we use variables for each changing activity $\bar{z}_a$ describing if connection $a \in \mathcal{A}_{change}$ is missed ($\bar{z}_a = 1$) or maintained ($\bar{z}_a = 0$). The idea of the activity-based formulation is to calculate the total delay by summing up the *additional delays* over all activities $a \in \mathcal{A}$. To this end, let us first consider some activity $a \in \mathcal{A} \setminus \mathcal{A}_{change}$. We want to calculate the additional delay customers will get while using this activity. The delay customers already have at the start of $a = (i,j)$ is y_i, and at the end of a their delay is y_j. Hence, $y_j - y_i$ is the additional delay gained by the customers while performing activity a. Note that this additional delay can be negative, meaning that slack times are used to compensate an already existing delay. For changing activities we have to be more careful. Let $a = (i,j) \in \mathcal{A}_{change}$ and suppose first that a is maintained. Then the additional delay on a is again the tension $y_j - y_i$. On the other hand, if

a is missed, the additional delay for the customers who planned to use activity a is given by $T - y_i = y_j - y_i + T - y_j$, since they now have to wait the remaining time period until the next (non-delayed) vehicle arrives for carrying on their journey.

We further need to extend the event-activity network by defining

$$\begin{aligned}\mathcal{E}^s &= \mathcal{E} \cup \{s\}\\ \mathcal{A}^s &= \mathcal{A} \cup \{(s,i) : i \in \mathcal{E}_{dep}\} \text{ and}\\ \mathcal{P}^s &= \{(s, i_1^p, \ldots, i_L^p) : p \in \mathcal{P}\}.\end{aligned}$$

The additional event s represents the arrival of the customers at their first station (by foot or by a means of transport which is not considered in the delay management problem). The extension makes sure that the delay of a customer waiting at some station for his first (delayed) vehicle to come, is taken into account. We always assume that customers reach their first station without any delay, i.e., $y_s = 0$.

Now we can present the new model. As before, we assume that $T, M \geq D$. The following additional variables are necessary for (TDM-C).

$$\tilde{z}_a^p = \begin{cases} 1 & \text{if activity } a \text{ is reached on path } p \text{ without any missed} \\ & \text{connection before} \\ 0 & \text{otherwise} \end{cases}$$

w_a = number of customers who *really* use activity a

We stress that the number of customers w_a (really) using activity $a \in \mathcal{A}$ is a variable, since it depends on the wait-depart decisions whether customers using a path $p \in \mathcal{P}^s$ will reach all activities $a \in p$ or not.

(TDM-C)

$$\min f_{\text{TDM-C}} = \sum_{a=(i,j)\in\mathcal{A}^s} w_a(y_j - y_i) + \sum_{a=(i,j)\in\mathcal{A}_{change}} w_a \bar{z}_a (T - y_j)$$

such that

$$y_i \geq d_i \quad \text{for all } i \in \mathcal{SD} \tag{11}$$

$$y_i - y_j \leq s_a \quad \text{for all } a = (i,j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \tag{12}$$

$$-M\bar{z}_a + y_i - y_j \leq s_a \quad \text{for all } a = (i,j) \in \mathcal{A}_{change} \tag{13}$$

$$\tilde{z}_a^p + \sum_{\substack{\tilde{a} \in p \cap \mathcal{A}_{change}: \\ \tilde{a} \prec a}} \bar{z}_{\tilde{a}} \geq 1 \quad \text{for all } p \in \mathcal{P}^s \text{ and } a \in p \tag{14}$$

$$\tilde{z}_a^p + \bar{z}_{\tilde{a}} \leq 1 \quad \text{for all } p \in \mathcal{P}^s \text{ and for all } a, \tilde{a} \in p \text{ with } \tilde{a} \in \mathcal{A}_{change} \text{ and } \tilde{a} \prec a \tag{15}$$

$$w_a = \sum_{p \in \mathcal{P}^s: a \in p} w_p \tilde{z}_a^p \quad \text{for all } a \in \mathcal{A}^s \tag{16}$$

$$y_i \in \mathbb{N} \quad \text{for all } i \in \mathcal{E} \tag{17}$$

$$\bar{z}_a \in \{0,1\} \quad \text{for all } a \in \mathcal{A}^s \tag{18}$$

$$\tilde{z}_a^p \in \{0,1\} \quad \text{for all } p \in \mathcal{P}^s, a \in \mathcal{A}^s \tag{19}$$

$$w_a \in \mathbb{N} \quad \text{for all } a \in \mathcal{A}^s \tag{20}$$

In the objective function the additional amount of delay on each activity is multiplied by the number of customers *really* using it. Restrictions (11) and (12) again correspond to (1) and (2), while (13) models that (3) has to be satisfied exactly for maintained connections, i.e. connections a with $\bar{z}_a = 0$. Restriction (14) defines the values of $\tilde{z}_a^p$ such that they are forced to be 1, if no connection on path p before a has been missed, and (15) makes sure that $\tilde{z}_a^p = 0$ for all activities a after a missed connection $\tilde{a}$ on path p. Finally, (16) determines the number of customers really using activity a.

Note that for technical reasons we need to be able to extend any feasible solution $y_i, i \in \mathcal{E}$ to a feasible solution $(y, C(y)) := (y, \bar{z}(y), \tilde{z}(\bar{z}), w(\tilde{z}))$ of (TDM-C), where $(y, C(y))$ yields the same or a better objective function value for (TDM-C). This is done as follows.

$$\bar{z}_a(y) = \begin{cases} 0 & \text{if } y_i - y_j \leq s_a \\ 1 & \text{otherwise} \end{cases} \quad \text{for all } a = (i,j) \in \mathcal{A}_{change}, \tag{21}$$

$$\tilde{z}_a^p(\bar{z}) = \max \left\{ 1 - \sum_{\substack{a \in p \cap \mathcal{A}_{change}: \\ \tilde{a} \prec a}} \bar{z}_{\tilde{a}}, 0 \right\} \quad \text{for all } p \in \mathcal{P}^s, a \in p, \tag{22}$$

$$w_a(\tilde{z}) = \sum_{p \in \mathcal{P}^s: a \in p} w_p \tilde{z}_a^p \quad \text{for all } a \in \mathcal{A}^s. \tag{23}$$

The main result of this section is the following.

Theorem 1. *(TDM-A) and (TDM-C) are equivalent. In particular, both models lead to the same set of optimal solutions $y \in \mathbb{R}^{|\mathcal{E}|}$.*

The proof can be found in the Appendix.

Using (TDM-C) we are able to derive the following reduction result. Assume that the slack times are so large that the delay disappears after a few activities. Then we need not consider events which can not gain any delay in the worst-case time-minimal solution.

Lemma 1. *Let $y = y(\mathcal{A}^{change})$ be a time-minimal solution w.r.t. $\mathcal{A}^{change}$. Then there exists an optimal solution $(y^*, \bar{z}^*, \tilde{z}^*, w^*)$ of (TDM-C) such that*

- *For all $i \in \mathcal{E}$: If $y_i = 0$ then $y_i^* = 0$.*
- *For all $a = (i,j) \in \mathcal{A}_{change}$: If $y_i = 0$ then $\bar{z}_a^* = 0$.*

The result shows that we need not consider events or activities which cannot gain a delay in the worst case. The *reduced set of events* is hence given as

$$\mathcal{E}_{relevant} = \{i \in \mathcal{E} : y_i(\mathcal{A}_{change}) > 0\}$$

and the subgraph induced by these events $\mathcal{E}_{relevant}$ is denoted by $\mathcal{N}_{relevant} = (\mathcal{E}_{relevant}, \mathcal{A}_{relevant})$. This kind of reduction leads to significantly smaller networks in real-world instances, see Table 1 in Section 5.

On a first glance, (TDM-C) does not seem to be useful for solving the delay management problem better than (TDM-B), since (TDM-B) is linear while (TDM-C) is cubic. Moreover, (TDM-C) is much larger in terms of variables, constraints, and non-zero entries of the coefficient matrix. However, it has some advantages. First, it is more general since it allows to replace the common time period T by time periods T_a for each changing activity $a \in \mathcal{A}_{change}$, which is a step to more realistic models and to relaxing our first assumption on page 7. Secondly, as the proof in the appendix shows, (TDM-C) is a stronger formulation than (TDM-A) and (TDM-B), since the decision variables $\bar{z}_a$ allow less freedom than the decision variables z_p. Hence, e.g., a classical branch-and-bound procedure using the variables $\bar{z}_a$ for branching can be easily implemented for (TDM-C) while for (TDM-A) other methods, e.g., constraint branching have to be investigated. Last, in the next section we utilize (TDM-C) to present a linear-time algorithm which solves the delay management problem exactly for a special class of problems.

4 Constant weights and the never-meet property

In order to solve (TDM-C) we fix the weights w_a as parameters instead of calculating them during the optimization. Doing so, we obtain the *total delay management problem with constant weights*. Its formulation is given by deleting constraints (14), (15), and (16) in (TDM-C), and fixing

$$w_a = \sum_{p \in \mathcal{P}^s : a \in p} w_p \quad \text{for all } a \in \mathcal{A}^s \tag{24}$$

as parameters, i.e., setting w_a as the planned "traffic load" on activity a. We obtain:

$$\min f_{\text{TDM}-\text{const}'} = \sum_{a=(i,j)\in\mathcal{A}^s} w_a(y_j - y_i) + \sum_{a=(i,j)\in\mathcal{A}_{change}} w_a \bar{z}_a(T - y_j)$$

such that (11),(12),(13),(17), and (18) hold.

We can further rewrite $f_{\text{TDM}-\text{const}'}$ as follows. For $i \in \mathcal{E}$ let

$$w_i = \sum_{p\in\mathcal{P}:i(p)=i} w_p \tag{25}$$

be the number of customers with final destination i. Since

$$\begin{aligned}\sum_{a=(i,j)\in\mathcal{A}^s} w_a(y_j - y_i) &= \sum_{p\in\mathcal{P}^s} w_p \sum_{a=(i,j)\in p} y_j - y_i \\ &= \sum_{p\in\mathcal{P}} w_p(y_{i(p)} - y_s) \\ &= \sum_{i\in\mathcal{E}} \sum_{\substack{p\in\mathcal{P}:\\ i(p)=i}} w_p y_i = \sum_{i\in\mathcal{E}} w_i y_i \end{aligned} \tag{26}$$

we rewrite

$$f_{\text{TDM}-\text{const}'} = \sum_{i\in\mathcal{E}} w_i y_i + \sum_{a=(i,j)\in\mathcal{A}_{change}} w_a \bar{z}_a(T - y_j).$$

First, we show that in general, we make a mistake by fixing the weights as above, which has not been realized in several previous attempts or simulation approaches for the delay management problem.

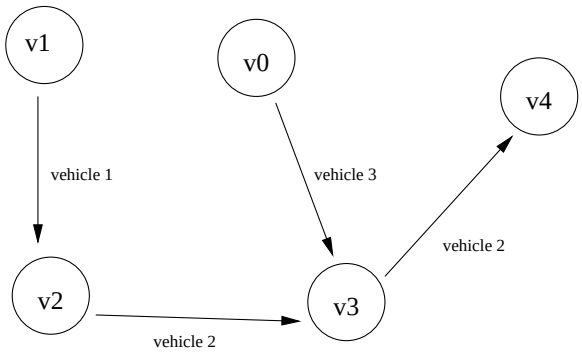

Fig. 3. An example in which fixing the weights is not correct.

We assume there are three vehicles 1, 2, and 3, where vehicle 1 and vehicle 3 reach the stations v_2 and v_3 with a delay, see Figure 3. We consider a customers' path $p = (v_1, v_2, v_3, v_4)$ using vehicle 1 until station v_2, changing to vehicle 2 and passing via v_3 to its destination v_4. Suppose that vehicle 2 is not waiting for vehicle 1 at station v_2, such that the path p is not maintained. Assume further that vehicle 2 waits for the delayed vehicle 3 at station v_3. If we have not adapted the weights, the customers on path p are counted twice in the objective function: First, since they missed their connection at station v_2, and secondly, since they reach their final destination v_4 with a delay. This double counting can in general lead to wrong decisions. Another example, depicted in Figure 5 will be further analyzed in Section 5.

Fortunately, there are problem instances for which the model with constant weights is correct, apart from the trivial case in which no customer changes at all. For example, it can be shown that the model with constant weights is correct, if we only allow paths of the form $p = (i_1, i_2, \ldots, i_{L-2}, i_{L-1}, i_L)$ where p contains at most one changing activity (i_{L-2}, i_{L-1}) followed by not more than one driving activity, see [Sch06]. A more interesting case, in which we make no mistake by using the constant weights will be described next.

Since $f_{\mathrm{TDM-const}'}$ still is no linear function we further simplify the model. In the following we simply forget about subtracting y_j in the second part of the objective, to obtain the **linear** program **(TDM-const)**.

$$\min f_{\mathrm{TDM-const}} = \sum_{i \in \mathcal{E}} w_i y_i + \sum_{a \in \mathcal{A}_{change}} w_a T \bar{z}_a$$

such that

$$\begin{aligned} y_i &\geq d_i \quad \text{for all } i \in \mathcal{SD} && (27)\\ y_i - y_j &\leq s_a \quad \text{for all } a = (i,j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive} && (28)\\ -M\bar{z}_a + y_i - y_j &\leq s_a \quad \text{for all } a = (i,j) \in \mathcal{A}_{change} && (29)\\ y_i &\in \mathbb{N} \ \ \forall i \in \mathcal{E}\\ \bar{z}_a &\in \{0,1\} \ \ \text{for all } a \in \mathcal{A}_{change} \end{aligned}$$

Each feasible solution of (TDM-const) yields an upper bound on (TDM). But the main advantage of (TDM-const) is due to the surprising fact that (TDM-const) is equivalent to (TDM) in a large class of practical instances. We denote

$$\mathcal{E}(i) = \{j \in \mathcal{E} : \text{ there exists a (directed) path from } i \text{ to } j\}$$

as the set of all events that can be reached from i, and $\mathcal{N}(i)$ as the subgraph induced by the events in $\mathcal{E}(i)$. Note that for all $j \in \mathcal{E}(i)$ we have $i \preceq j$. Furthermore, let $\mathcal{E}(\mathcal{SD}) = \bigcup_{i \in \mathcal{SD}} \mathcal{E}(i)$ and $\mathcal{N}(\mathcal{SD})$ be the subgraph containing the subgraphs $\mathcal{N}(i)$ for all $i \in \mathcal{SD}$, i.e. the graph consisting of all events and activities that can be reached by a path starting at a source-delayed event. Obviously, $\mathcal{E}_{relevant} \subseteq \mathcal{E}(\mathcal{SD})$.

Definition 4. *The delay management problem has the* **never-meet property** *if the following two conditions hold.*

1. $\mathcal{N}(i) \cap \mathcal{N}_{relevant}$ *is a forest for all* $i \in \mathcal{SD}$*, and*
2. $\mathcal{E}(i) \cap \mathcal{E}(j) \cap \mathcal{E}_{relevant} = \emptyset$ *for all* $i, j \in \mathcal{SD}$ *with* $i \neq j$*.*

Note that $\mathcal{N}(\mathcal{SD}) \cap \mathcal{N}_{relevant}$ is a forest, whenever the never-meet property holds, i.e. it is not allowed to contain cycles (neither directed nor undirected cycles).

The interpretation of the never-meet property is the following: By calculating the time-minimal solution (w.r.t. $\mathcal{A}^{fix} = \mathcal{A}_{change}$), but without using slack-times, we can find out how far the effects of the source delays can spread out in the worst case. The never-meet property requires that in **no** feasible solution of (TDM) the paths of two delayed customers will meet. Note that the formulation includes that source delays can only occur after non-delayed events.

If the never-meet property holds, however, we will show the following: In every time-minimal solution all events following a non-maintained connection are punctual, and all changing activities following a non-maintained connection are maintained. This property will be important for proving Theorem 2.

Lemma 2. *Let (TDM) have the never-meet property and let* $(y, C(y))$ *be a feasible (time-minimal) solution of (TDM-C). Let* $\tilde{a} = (\tilde{i}, \tilde{j}) \in \mathcal{A}_{change}$*. If* $\bar{z}_{\tilde{a}} = 1$ *(i.e.* $\tilde{a}$ *is not maintained) we have the following.*

1. $y_i = 0$ *for all* $i \in \mathcal{E}(\tilde{j})$*, i.e. all events following* $\tilde{j}$ *are on time, and*
2. $\bar{z}_a = 0$ *for all* $a = (i, j)$ *with* $i \in \mathcal{E}(\tilde{j})$*, i.e., all connections following* $\tilde{j}$ *are maintained.*

Proof. From $\bar{z}_{\tilde{a}} = 1$ we know from (21) that $y_{\tilde{i}} > 0$. Hence there exists a source-delayed event $i_1 \in \mathcal{SD}$ such that $\tilde{i} \in \mathcal{E}(i_1)$. Now suppose there exists $i \in \mathcal{E}(\tilde{j}) \subseteq \mathcal{E}(i_1)$ with $y_i > 0$. Since $\bar{z}_{\tilde{a}} = 1$ the delay of i is not transferred from i_1 to i via $\tilde{a}$. Hence

- either there is another path from i_1 to i, meaning that $\mathcal{N}(i_1) \cap \mathcal{N}_{relevant}$ is not a tree, or
- the delay of y_i is caused by another source-delayed event $i_2 \in \mathcal{E}$, meaning that $i \in \mathcal{E}(i_1) \cap \mathcal{E}(i_2) \cap \mathcal{E}_{relevant}$.

In both cases we have a contradiction to the never-meet property. Finally, consider $a = (i, j)$ with $i, j \in \mathcal{E}(\tilde{j})$. From part 1 we know that $y_i = y_j = 0$, hence (21) yields $\bar{z}_a = 0$. □

We can now present our main result.

Theorem 2. *Model (TDM-const) is correct if the never-meet property holds.*

Proof. We show that (TDM-C) and (TDM-const) are equivalent if the never-meet property holds. Clearly, a feasible solution $(y, \bar{z})$ of (TDM-const) can be extended to a feasible solution $(y, C(y))$ of (TDM-C) with equal or better objective value, see (21), (22), and (23).

The other direction is the interesting one: We show that each feasible solution of (TDM-C) corresponds to a feasible solution of (TDM-const) with the same or better objective value. More precisely, given some feasible solution of (TDM-C) with delay y, let $(y, C(y)) = (y, \bar{z}, \tilde{z}, w^{real})$ denote a (maybe better) feasible solution of (TDM-C). We show that $(y, \bar{z})$ is a feasible solution of (TDM-const) with the same objective value as $(y, C(y))$. Feasibility of $y, \bar{z}$ for (TDM-const) is trivially satisfied. It remains to show that

$$f_{\mathrm{TDM-C}}(y, \bar{z}, \tilde{z}, w) = f_{\mathrm{TDM-const}}(y, \bar{z}).$$

To this end, suppose that for some $\bar{a} = (\bar{i}, \bar{j}) \in \mathcal{A}$ we made a mistake by fixing the weights, i.e., the number of customers $w_{\bar{a}}$ who planned to use $\bar{a}$ does not equal the number of customers $w_{\bar{a}}^{real}$, really using $\bar{a}$. To compare the objective functions of (TDM-const) and (TDM-C) we replace the first term of (TDM-const) by equation (26) and see that in this case it suffices to show that

$$y_{\bar{j}} - y_{\bar{i}} = 0,$$

and that, if $\bar{a} \in \mathcal{A}_{change}$

$$z_{\bar{a}} = 0,$$

This means that the error we make by using the wrong weights does not influence the value of the objective function. From $w_{\bar{a}} \neq w_{\bar{a}}^{real}$ we get (by comparing (23) and (24)), that

$$\sum_{p \in \mathcal{P}^s : \bar{a} \in p} w_p = w_{\bar{a}} \neq w_{\bar{a}}^{real} = \sum_{p \in \mathcal{P}^s : \bar{a} \in p} w_p \tilde{z}_{\bar{a}}^p.$$

Hence there exists some path $p \in \mathcal{P}$ containing $\bar{a}$ such that $\tilde{z}_{\bar{a}}^p = 0$. Due to (22) there exists $\tilde{a} \in p$ with $\tilde{a} \prec \bar{a}$ and $\bar{z}_{\tilde{a}} = 1$. Without loss of generality let us take $\tilde{a} = (\tilde{i}, \tilde{j})$ minimal with this property, i.e., we choose the first changing activity on path p that is marked as missed. For an illustration, see Figure 4.

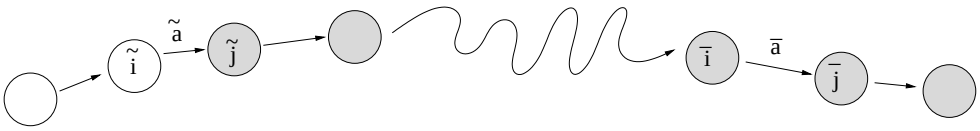

Fig. 4. The path p in the proof of Theorem 2. The grey events belong to $\mathcal{E}(\tilde{j})$.

Since $\bar{i}, \bar{j} \in \mathcal{E}(\tilde{j})$ we derive from Lemma 2 that

- $y_{\bar{i}} = y_{\bar{j}} = 0$, and
- if $\bar{a} \in \mathcal{A}_{change}$ then $\bar{z}_a = 0$.

Hence, $y_{\bar{j}} - y_{\bar{i}} = 0$, and if $\bar{a} \in \mathcal{A}_{change}$ we have that $z_{\bar{a}} = 0$, which completes the proof. □

5 The never-meet property in practice

To investigate the never-meet property in practice, we used real-world data of a part of the German railway network, namely around the region of Harz in Germany. The data we used consists of 158 railway stations, and 1101 different trains running at one particular day.

Since the never-meet property does not depend on the set of paths $\mathcal{P}$, we used two different sets $\mathcal{U}_{30}$, and $\mathcal{U}_{60}$ to generate potential connections. The set $\mathcal{U}_{30}$ contains reasonable connections within a scheduled waiting time between 3 and 30 minutes, while we allow a waiting time between 3 and 60 minutes in $\mathcal{U}_{60}$. By "reasonable" we mean that we do not consider connections where a transfer results in going directly back to the previous station. The size of U_{30} is 5567, while U_{60} contains 11229 connections. The resulting event-activity network of the public transportation network on our particular day has a size of 10492 events. The number of activities on this day depends on the allowed transfer time and varies between 13359 and 17616. In our numerical study we generated 500 example sets of delays with up to 5 source delays.

As shown in Table 1 the event-activity network can be drastically reduced if we delete all events that can never gain a delay. The table demonstrates the results of the reduction for different numbers of source delays. Each row contains the average number of events for 100 different delay scenarios. In column $|\mathcal{E}(\mathcal{SD})|$ the number of events that can be reached by a path from one of the source delays is given, while column $|\mathcal{E}(\mathcal{SD}) \cap \mathcal{E}_{relevant}|$ shows how many of these events can gain a delay and hence have to be considered in the optimization. The percentage of reduction is given in the last column.

Table 1. Reduction of the original set of 10492 events for different delay scenarios in the case of a transfer time up to 30 minutes.

no. of source delays	$\lvert\mathcal{E}(\mathcal{SD})\rvert$	$\lvert\mathcal{E}(\mathcal{SD}) \cap \mathcal{E}_{relevant}\rvert$	reduction to
1	2668	228	8.5%
2	4172	460	11.0%
3	5029	599	11.9%
4	5344	801	15.0%
5	5495	948	17.3%

The never-meet property cannot be sharpened by further reducing the sets $\mathcal{E}(i) \cap \mathcal{E}_{relevant}$ and only looking at the smaller sets $\mathcal{E}'(i) \subseteq \mathcal{E}(i) \cap \mathcal{E}_{relevant}$ containing only those events that can gain a delay originating at $i \in \mathcal{SD}$. This is demonstrated in the following example with two source delays at events v_1 and v_2. Source delay 1 spreads out to the dashed nodes $\mathcal{E}'(v_1)$, while the grey nodes (denoted by $\mathcal{E}'(v_2)$) show which events can gain a delay from source delay 2. Suppose that – due to sufficiently large slack times – no other events can

be affected. Although the dashed and the grey nodes form trees, and their intersection is empty, the example does not have the never-meet property (since $v_4 \in \mathcal{E}(v_1) \cap \mathcal{E}(v_2) \cap \mathcal{E}_{relevant}$)! But this is what we want, since Theorem 2 does also not hold in this example: Consider path P from v_3 to v_4. Assume that a is a changing activity and it is missed. Then customers traveling along P never reach node v_4 and would be (wrongly) counted there.

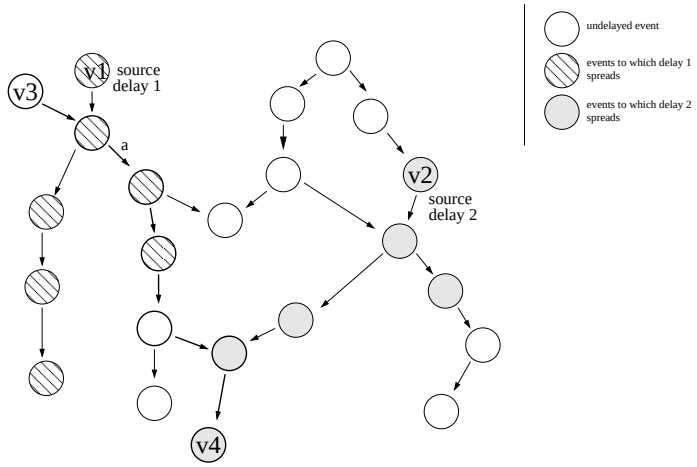

Fig. 5. Although source delay 1 disappeared before reaching an event which has a delay coming from source delay 2, this example does not have the never-meet property. In particular, Theorem 2 is not true in this example.

We tested the never-meet property in practice, which can be done efficiently by the forward phase of the critical path method (with zero slack times and $\mathcal{A}^{fix} = \mathcal{A}_{change}$). To analyze the results, let us call an event i *in conflict with the never-meet property*, if it can be reached by more than one path originating in a source-delayed event. The number of all events which are in conflict with the never-meet property is called the number of *node conflicts* of the problem. The events which are in conflict with the never-meet property can be determined by looking at their in-degrees within the graph $\mathcal{N}(\mathcal{SD})$. More precisely,

- an event $i \in \mathcal{E}_{relevant} \setminus \mathcal{SD}$ is in conflict with the never-meet property, if its in-degree in the graph $\mathcal{N}(\mathcal{SD})$ is at least 2. The in-degree minus 1 is called its *degree of conflicts*.
- Event $i \in \mathcal{SD}$ is in conflict with the never-meet property, if its in-degree in the graph $\mathcal{N}(\mathcal{SD})$ is at least 1. In this case, its in-degree equals its *degree of conflicts*.

The sum of all degrees of conflict will be called the number of *edge conflicts* with the never-meet property. It equals the number of edges which have to be deleted to ensure that the never-meet property holds.

In Figure 6 the number of conflicts with the never-meet property (node and edge conflicts) is depicted as a function of the number of delayed vehicles. Note that we considered scenarios with 1,2,3,4, and 5 source delays and generated 100 examples for each of these scenarios, with different amounts of source delay. Each example is given by a "+" in the figure. The average values for each amount of source delay are given by circles. As expected, the average number of conflicts with the never-meet property is relatively small for only one source delay, while it increases when more than one source delay is considered. Furthermore, the variance increases with the number of source delays: There are still many examples with 5 source delays which lead to very few conflicts, but there are also examples with many conflicts.

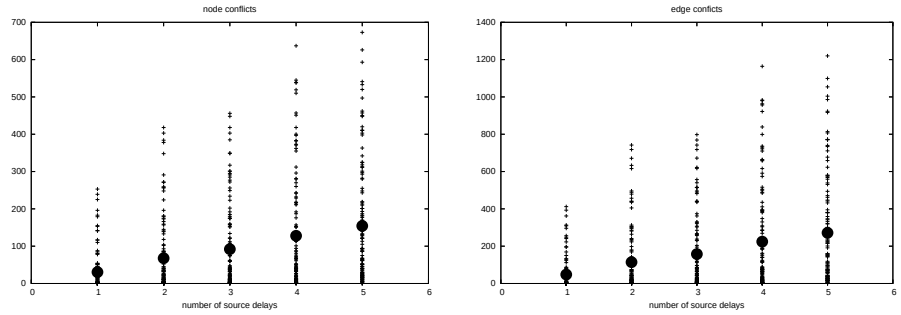

Fig. 6. The number of node conflicts (left) and edge conflicts (right) as a function of the number of source delays.

Figure 7 shows the same data, but here we graph the number of edge and node conflicts with the never-meet property as a function of the *amount* of the source delays. We observe that the number of conflicts increases if the source delays increase, and that small source delays are very likely to generate nearly no conflicts with the never-meet property.

The reason for the relatively small number of conflicts in practice is in particular due to the fact that we only consider events in $\mathcal{E}_{relevant}$, i.e. only events that can gain a delay. As expected, most conflicts with the never-meet property arise at the larger stations, while the never-meet property is more likely to hold for smaller stations in a rural environment. But all this is only helpful if we can draw advantage of the simplified model with constant weights in terms of efficiently solving it. This will be investigated in the next section.

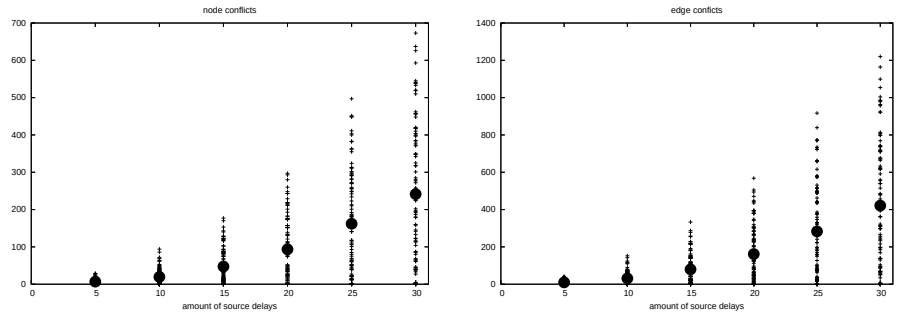

Fig. 7. The number of node conflicts (left) and edge conflicts (right) as a function of the amount of delay.

6 Solving (TDM-const)

The main goal of this section is to solve (TDM) in case of the never-meet property. Using Theorem 2 it is enough in this case to develop an algorithm for (TDM-const).

The first approach is to just use an integer programming solver. We solved our example problems using GLPK (GNU linear programming kit). The results for the examples with one source delay are illustrated in Figure 8. As before, each "+" refers to one example. The first coordinate shows the the number of conflicts with the never-meet property (node and edge conflicts, respectively), while the second coordinate represents the number of pivot operations needed for solving the corresponding program (TDM-const) to optimality. We observe that the number of pivot operations for solving (TDM-const) increases with the number of conflicts with the never-meet property, but not as badly as one could have expected.

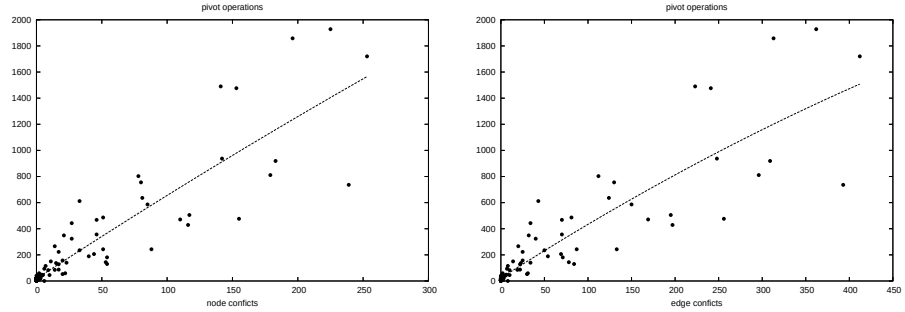

Fig. 8. The number of pivot operations needed for solving (TDM-const) versus the number of node conflicts (left) and edge conflicts (right).

To understand the reason for this behavior we first look at the following special case of (TDM) with the never-meet property, in which

- all source delays have the same amount, i.e., $d_i \in \{0, D\}$ for all $i \in \mathcal{E}$, and
- all slack times are equal to zero, i.e., $s_a = 0$ for all $a \in \mathcal{A}$.

Let y be a time-minimal solution of this problem. Then $y_i \in \{0, D\}$ for all $i \in \mathcal{E}$. This means that we can use binary variables y_i instead of integer ones, with

$$y_i = \begin{cases} 1 & \text{if event } i \text{ is delayed by } D \\ 0 & \text{if event } i \text{ is not delayed.} \end{cases}$$

Consequently, $M = 1$ is large enough and (TDM-const), even with the first objective $f_{\mathrm{TDM-const}'}$ introduced on page 12, simplifies to the following **linear** program.

(TDM-const-zero)

$$\min \sum_{a=(i,j)\in\mathcal{A}^s} w_a D(y_j - y_i) + \sum_{a=(i,j)\in\mathcal{A}_{change}} w_a \bar{z}_a (T - D)$$

such that

$$\begin{aligned} -y_i &\le -1 \quad \text{for all } i \in \mathcal{SD} && (30)\\ y_i - y_j &\le 0 \quad \text{for all } a = (i,j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive} && (31)\\ \bar{z}_a + y_i - y_j &\le 0 \quad \text{for all } a = (i,j) \in \mathcal{A}_{change} && (32)\\ y_i &\in \{0,1\} \quad \forall i \in \mathcal{E} \\ \bar{z}_a &\in \{0,1\} \quad \forall a \in \mathcal{A}_{change}, \end{aligned}$$

where $w_a = \sum_{p\in\mathcal{P}^s : a\in p} w_p$ for all $a \in \mathcal{A}^s$ are given parameters as before (see, e.g., (24)). The following result explains the good behavior of mixed integer programming for (TDM-const).

Theorem 3. *The coefficient matrix of (TDM-const-zero) is totally unimodular.*

Proof. Let $C = |\mathcal{A}_{change}|$, $\bar{C} = |\mathcal{A}_{drive} \cup \mathcal{A}_{wait}|$ and $\bar{D} = |\mathcal{SD}|$. Moreover, let I_K denote the unit matrix of size $K \times K$ and $O_{K,L}$ the zero matrix of size $K \times L$. Then the coefficient matrix of (TDM-const-zero) is

$$\Phi = \left(\begin{array}{c|c} -I_{\bar{D}} & 0_{\bar{D},C} \\ \hline \Theta^T & \begin{array}{c} 0_{\bar{C},C} \\ \hline I_C \end{array} \end{array}\right),$$

where the $|\mathcal{A}| \times |\mathcal{E}|$-matrix Θ^T is the transposed of the node-arc-incidence matrix Θ of $\mathcal{N}$, and hence totally unimodular. Consequently, Φ is also totally unimodular. □

We remark that (TDM-const-zero) is equivalent to the models developed independently in diploma theses by Kliewer [Kli00] and Scholl [Sch01a], where the latter author also recognized the total unimodularity of the model.

The lemma gives the explanation for the graphics of Figure 8: In case of zero slack times the LP-relaxation of (TDM-const-zero) yields an integer solution (see e.g., [NW88]), which makes the problem efficiently solvable in this case. Since the structure of the problem does not change by introducing slack times, one can hope for an efficient algorithm also in the general case. We therefore turn our attention back to the case of non-zero slack times and will develop an efficient algorithm (running in $O(|\mathcal{A}|)$ time) for solving (TDM-const). In contrast to the case with zero slack times, this approach relies on the never-meet property. In particular we will utilize the two facts listed below.

- First, if we fix $\bar{z}_a = 1$ for some $a = (i, j)$, we can set $y_{i'} = 0$ for all $i' \in \mathcal{E}(j)$ and know that all subsequent connections are maintained (Lemma 2).
- Secondly, the problem can be decomposed into at most $|\mathcal{A}_{change}|$ independent subproblems due to the following lemma, which also follows directly from the never-meet property.

Lemma 3. *Let $i, j \in \mathcal{E}$, $i \neq j$, and let $(y, \bar{z})$ be a feasible solution of (TDM-const) with $y_i > 0, y_j > 0$. If the never-meet property holds, exactly one of the following three cases occurs.*

$$\mathcal{E}(i) \subseteq \mathcal{E}(j) \quad or \quad \mathcal{E}(j) \subseteq \mathcal{E}(i) \quad or \quad \mathcal{E}(i) \cap \mathcal{E}(j) = \emptyset.$$

The idea of the algorithm is to decompose the problem iteratively into subproblems, and solve them bottom-up. A subproblem P_a is identified by a changing activity $a = (i, j)$ and represents the delay management problem on the subgraph $\mathcal{N}(i)$ (recall the notation on page 13) with a single source delay at event i. P_a might be decomposable into subproblems itself. Formally, we define

$$\mathrm{SP}(a) = \{a' \in \mathcal{A}_{change} : \text{ there exists a directed path from } a \text{ to } a' \text{ not containing any other changing activity}\}$$

The subproblems of the problem itself are collected in $\mathrm{SP}(a_0)$, and can be derived by taking all changing activities reachable directly from one of the source-delayed events.

We remark that all subproblems within the same set $\mathrm{SP}(a)$ are independent of each other due to the never-meet property.

In Algorithm 2, subproblems that might further be decomposed are stored in "Decompose", and if a subproblem cannot be decomposed any more it is collected in "Compose". Moreover, at the end of Step 2 of the algorithm, for each subproblem identified by some changing activity a,

- maintain(a) contains the value of the objective function of the subproblem if a is maintained, and

- miss(a) contains the objective value if a is missed.
- $f(a)$ contains the minimum of maintain(a) and miss(a).

To compute maintain(a) we need to calculate the minimum delay which occurs if a is maintained. In contrast to $\mathcal{E}(i)$ which is the set of events that can be reached from i, if *all* $a \in \mathcal{A}_{change}$ can be used we now define

$$\mathcal{G}(i) = \{j \in \mathcal{E} : \text{ there exists a path from } i \text{ to } j \text{ with activities in } \mathcal{A}_{wait} \cup \mathcal{A}_{drive}\}$$

as the set of events that can be reached from i without passing any changing activity. Furthermore, assume that $i \in \mathcal{E}$ has a delay $d_i > 0$, and let y be a time-minimal solution. The minimum delay that will be caused by d_i independent of any wait-depart decision is then given by

$$G(i, d_i) = \sum_{j \in \mathcal{G}(i)} w_j y_j.$$

The algorithm can now be stated.

Algorithm 2: Enumeration for (TDM-const)

Input: $\mathcal{N}, \mathcal{P}, w_p, d_i, s_a, T$.
Output: Optimal solution of (TDM), if the never-meet property holds.
Step 0.

1. Calculate the time-minimal solution $y(\mathcal{A}_{change})$ if all connections are maintained by Algorithm 1.
2. (Initializations) Let a_0 denote (TDM-const), set $\mathrm{SP}(a_0) = \emptyset$, $f(a_0) = 0$, $\mathrm{Decompose} = \emptyset$, $\mathrm{Compose} = \emptyset$, $\bar{z}_a = 0$ for all $a \in \mathcal{A}_{change}$.
3. (Calculate $\mathrm{SP}(a_0)$) For all $i \in \mathcal{SD}$:
   (a) $f(a_0) = f(a_0) + G(i, d_i)$
   (b) For all $a = (j_1, j_2) \in \mathcal{A}_{change}$ with $j_1 \in \mathcal{G}(i)$: If $y_{j_1} > 0$ then $\mathrm{SP}(a_0) = \mathrm{SP}(a_0) \cup \{a\}$, and $\mathrm{Decompose} = \mathrm{Decompose} \cup \{a\}$
4. (Optimality test) If $\mathrm{SP}(a_0) = \emptyset$ stop: f is the optimal objective value, $\bar{z}_a = 0$ for all $a \in \mathcal{A}_{change}$

Step 1. While $\mathrm{Decompose} \neq \emptyset$

1. Choose $a = (i_1, i_2) \in \mathrm{Decompose}$
2. $\mathrm{SP}(a) = \emptyset$, $\mathrm{miss}(a) = w_a T$, $\mathrm{maintain}(a) = G(i_2, y_{i_2})$
3. (Calculate $\mathrm{SP}(a)$) For all $a' = (j_1, j_2) \in \mathcal{A}_{change}$ with $j_1 \in \mathcal{G}(i_2)$: If $y_{j_1} > 0$ then $\mathrm{SP}(a) = \mathrm{SP}(a) \cup \{a'\}$, $\mathrm{Decompose} = \mathrm{Decompose} \cup \{a'\}$
4. (Update Compose) If $\mathrm{SP}(a) = \emptyset$ then $\mathrm{Compose} = \mathrm{Compose} \cup \{a\}$.
5. (Update Decompose) $\mathrm{Decompose} = \mathrm{Decompose} \setminus \{a\}$.

Step 2. While $\mathrm{Compose} \neq \emptyset$.

1. Choose $a \in \mathrm{Compose}$. Let $\tilde{a}$ be parent of a, i.e. $a \in \mathrm{SP}(\tilde{a})$
2. (Solve subproblem P_a) $f(a) = \min\{\mathrm{maintain}(a), \mathrm{miss}(a)\}$,

$$\bar{z}_a = \begin{cases} 0 & \text{if } \mathrm{maintain}(a) \leq \mathrm{miss}(a) \\ 1 & \text{if } \mathrm{maintain}(a) > \mathrm{miss}(a) \end{cases}$$

3. (Update values for parent problem $\tilde{a}$)
   $\mathrm{SP}(\tilde{a}) = \mathrm{SP}(\tilde{a}) \setminus \{a\}$, $\mathrm{maintain}(\tilde{a}) = \mathrm{maintain}(\tilde{a}) + f(a)$

4. `(Update` Compose)
   Compose = Compose $\setminus \{a\}$
   `If` $\mathrm{SP}(\tilde{a}) = \emptyset$ `and` $\tilde{a} \neq a_0$ `then` Compose = Compose $\cup \{\tilde{a}\}$

Step 3.

1. `(Correct values for` $\bar{z}_a$`) For all` $a \in \mathcal{A}_{change}$`: If` $\bar{z}_a = 1$ `then set` $\bar{z}_{a'} = 0$ `for all` $a' \neq a$ `with` $a \prec a'$.
2. `Output:` $f(a_0) :=$ maintain(a_0), $\bar{z}$

Theorem 4. *Algorithm 2 is correct and runs in time O(|$\mathcal{A}$|).*

Proof. We show by induction over all $a \in \mathcal{A}_{change} \cup \{a_0\}$ that $f(a)$ contains the objective value for the subproblem P_a at the end of Algorithm 2.

Start: Let $a = (i, j)$ be a maximal element of $\mathcal{A}_{change}$ (with respect to $\prec$). The subproblem with respect to a is (TDM-const) in the small network $\mathcal{N}(i)$, which does not contain any changing activity except a itself, hence $\mathrm{SP}(a) = \emptyset$ in step 2 of the algorithm. Furthermore,

$$\text{maintain}(a) = \sum_{i' \in \mathcal{G}(j)} y_{i'} w_{i'}, \text{ and}$$
$$\text{miss}(a) = T w_a$$

give the objective values of this small network when maintaining or not maintaining activity a. To see the correctness of miss(a) we note that due to Lemma 2 $y_{i'} = 0$ for all $i' \in \mathcal{E}(j)$ (which equals $\mathcal{G}(j)$ in this case).
Since $a \in$ Compose we compare both values maintain(a) and miss(a) in step 2, and choose the better as (correct) objective value, which is then stored in $f(a)$.

Conclusion: Now take any $a = (i, j)$ and let the induction hypothesis be true for all a' with $a \prec a'$.

- If a is not maintained, we know from Lemma 2 that all connections $a' \in \mathcal{N}(j)$ are maintained and all $i' \in \mathcal{E}(j)$ satisfy $y_{i'} = 0$, i.e., the objective value is given by miss(a) as calculated in step 2.
- If a is maintained, the algorithm calculates in step 2 the delay which will be gained in any case, i.e., the delay of all events $i' \in \mathcal{G}(i)$ that can be reached without passing any changing activity, and store it in maintain(a). All changing activities a' that can be reached from j without passing any other changing activity are stored in $\mathrm{SP}(a)$. Due to Lemma 3 the corresponding subproblems $P_{a'}$ for $a' \in \mathrm{SP}(a)$ are independent and have objective value $f(a')$ due to the induction hypothesis, such that maintain$(a) + \sum_{a' \in \mathrm{SP}(a)} f(a')$ calculated in step 2.3 finally is the correct value of maintain.

Comparing maintain(a) with miss(a) and choosing the smaller of both gives the best possible choice for activity a assuming the delay y_i as given.

Finally, in step 0, the problem with the given source delays is decomposed into a set of subproblems $\mathrm{SP}(a_0)$. All these subproblems are independent due to Lemma 3, and they are all solved optimally due to the claim above. Adding up these optimal values and adding the delay of all events which are reached before entering one of the subproblems gives the optimal objective function value $f(a_0)$.

For the time complexity we note that the number of subproblems equals the number of changing activities, which in a tree is the same as the number of events. For the decomposition steps we have to process each activity and each event exactly once, and in the composition step we need one comparison and one summation for each subproblem, and again a visit of all events. The overall time complexity is hence linear in $|\mathcal{A}|$. □

7 Future Research

Algorithm 2 relies on the fact that each activity $a \in \mathcal{A}_{change}$ appears in exactly one list, i.e., for each $a \in \mathcal{A}_{change}$ there exists a unique $\tilde{a}$ such that $a \in \mathrm{SP}(\tilde{a})$, or $a \in \mathrm{SP}(a_0)$. If the never-meet property is not satisfied, this needs not be the case, and hence Algorithm 2 cannot be applied to (TDM) for general problems. To resolve this problem (and to obtain a heuristic by applying Algorithm 2) one can either allow that the same element is added more than once to *Compose* in step 2 (this would mean to duplicate activities until the never-meet property is satisfied), or to update the values of *maintain* to the larger one, if an element which is already contained is added.

(TDM-const) and (TDM) can both be solved by branch and bound, taking $\bar{z}_a$ as branching variables and reducing the number of conflicts with the never-meet property in each node. Lower bounds are derived in [Sch06]. Details and implementations are under research.

Two other directions of future research in delay management should be mentioned. First, it is a challenging task to apply delay management approaches in railway transportation. The drawback here is that capacity constraints have to be taken into account on the tracks. Different possibilities how such constraints can be included in the models are under research, see [BGJ+05]. Second, it is an open field to deal with the stochastic nature of the delays instead of assuming that the source delays are fixed.

Appendix

Proof of Theorem 1: (TDM-A) and (TDM-C) lead to the same set of optimal solutions $y \in \mathbb{R}^{|\mathcal{E}|}$.

Proof. First, using (16) the objective function of (TDM-C) can be reformulated to

$$\begin{aligned} f_{\text{TDM-C}} &= \sum_{a=(i,j)\in\mathcal{A}^s} w_a(y_j - y_i) + \sum_{a=(i,j)\in\mathcal{A}_{change}} w_a\bar{z}_a(T - y_j) \\ &= \sum_{a=(i,j)\in\mathcal{A}^s} \sum_{p\in\mathcal{P}^s:a\in p} w_p\tilde{z}_a^p(y_j - y_i) + \sum_{a=(i,j)\in\mathcal{A}_{change}} \sum_{p\in\mathcal{P}^s:a\in p} w_p\tilde{z}_a^p\bar{z}_a(T - y_j) \\ &= \sum_{p\in\mathcal{P}^s} w_p \left(\sum_{a=(i,j)\in\mathcal{A}^s:a\in p} \tilde{z}_a^p(y_j - y_i) + \sum_{\substack{a=(i,j)\in\mathcal{A}_{change} \\ a\in p}} \tilde{z}_a^p\bar{z}_a(T - y_j) \right) \\ &=: \sum_{p\in\mathcal{P}^s} w_p C_p. \end{aligned}$$

For the objective of (TDM-A), we define

$$A_p = y_{i(p)}(1 - z_p) + Tz_p.$$

(TDM-C) $\Longrightarrow$ (TDM-A): Let $(y, \bar{z}, \tilde{z}, w)$ be feasible for (TDM-C). Define $z_p = z_p(\bar{z})$ as follows:

$$z_p(\bar{z}) = \begin{cases} 0 & \text{if } \bar{z}_a = 0 \text{ for all } a \in p \cap \mathcal{A}_{change} \\ 1 & \text{otherwise} \end{cases} \tag{33}$$

Then (4) holds due to (11), (5) holds due to (12), and (6) is trivially satisfied, if $z_p = 1$, and for $z_p = 0$ we know that $\bar{z}_a = 0$ for all $a \in p$ and hence (6) holds because of (13). This means (y, z) is feasible for (TDM-A). It remains to show that $A_p \le C_p$. To this end, let $p = (s, i_1, \ldots, i_L) \in \mathcal{P}^s$ be a path with $i(p) = i_L$.

Case 1: $\bar{z}_a = 0$ for all $a \in p \cap \mathcal{A}_{change}$. Then, we define $z_p = 0$. From (14) we get that $\tilde{z}_a^p = 1$ for all $a \in p$. Since $y_s = 0$ we conclude that

$$C_p = \sum_{a=(i,j)\in\mathcal{A}^s:a\in p} y_j - y_i = y_{i_L} - y_s = A_p.$$

Case 2: There exists $a \in p \cap \mathcal{A}_{change}$ with $\bar{z}_a = 1$. Choose a minimal with respect to $\prec$ with this property, say $\bar{a} = (i_{\bar{k}-1}, i_{\bar{k}})$. Then, since $\bar{z}_a, \tilde{z}_a^p$ satisfy (14) and (15) we obtain

$$\begin{aligned} \tilde{z}_a^p &= 0 \text{ for all } a \in p \text{ with } \bar{a} \prec a \\ \tilde{z}_a^p &= 1 \text{ for all } a \in p \text{ with } a \preceq \bar{a}. \end{aligned}$$

Hence, for all $a \in \mathcal{A}_{change} \cap p$ we get

$$\tilde{z}_a^p\bar{z}_a = \begin{cases} 1 & \text{if } a = \bar{a} \\ 0 & \text{otherwise} \end{cases}$$

This yields

$$\begin{aligned} C_p &= \sum_{\substack{a=(i,j)\in\mathcal{A}^s: a\in p \\ \text{and } a\preceq\bar{a}}} y_j - y_i + (T - y_{i_{\bar{k}}}) \\ &= y_{i_{\bar{k}}} - y_s + T - y_{i_{\bar{k}}} = T = A_p, \end{aligned}$$

and consequently, $f_{\mathrm{TDM-C}}(y,\bar{z},\tilde{z},w) = f_{\mathrm{TDM-A}}(y,z(\bar{z}))$.

(TDM-A) $\Longrightarrow$ (TDM-C): Now let a feasible solution $(\tilde{y},z)$ of (TDM-A) be given. Using Algorithm 1 we may replace $\tilde{y}$ by a time-minimal solution y which satisfies $y_i \le T$ for all $i \in \mathcal{E}$, and has equal or better objective value. Since y satisfies (4) and (5) we can construct a feasible solution for (TDM-C) according to (21),(22), and (23).

For the objective value of this solution we again compare C_p and A_p for a path $p = (s, i_1, \ldots, i_L) \in \mathcal{P}^s$ and get:

Case 1: If $z_p = 0$, we get from (6) that $y_i - y_j \le s_a$ for all $a = (i,j) \in p$. Hence, due to the definition of $\bar{z}_a$ we conclude that $\bar{z}_a = 0$ for all $a \in p \cap \mathcal{A}_{change}$, yielding $C_p = y_{i(p)} = A_p$ analogously to Case 1 of the first part of the proof.

Case 2: Now consider the case $z_p = 1$.

Case 2a: $y_i - y_j \le s_a$ for all $a = (i,j) \in p$, yielding that $\bar{z}_a = 0$ for all $a \in p$ and hence $C_p = y_{i(p)} \le T = A_p$.

Case 2b: There exists $a = (i,j) \in p$ such that $y_i - y_j > s_a$. This gives us $\bar{z}_a = 1$. Choose $\bar{a} = (i_{\bar{k}-1}, i_{\bar{k}})$ minimal with respect to $\prec$ with this property. Then, from the definition of $\tilde{z}_a^p$ we get

$$\begin{aligned} \tilde{z}_a^p &= 0 \text{ for all } a \in p \text{ with } \bar{a} \prec a \\ \tilde{z}_a^p &= 1 \text{ for all } a \in p \text{ with } a \preceq \bar{a} \end{aligned}$$

and analogously to Case 2 of the first part of the proof $C_p = T = A_p$.

Together, $f_{\mathrm{TDM-A}}(\tilde{y},z) \ge f_{\mathrm{TDM-A}}(y,z) \ge f_{\mathrm{TDM-C}}(y,C(y))$.

Combining both directions yields that there exists an optimal solution for (TDM-A) with delays y if and only if there exists an an optimal solution for (TDM-C) with the same delays y. □

References

[Ack99] T. Ackermann. *Die Bewertung der Pünktlichkeit als Qualitätsparameter im Schienenpersonennahverkehr auf Basis der direkten Nutzenmessung.* PhD thesis, Universität Stuttgart, 1999.

[APW02] L. Anderegg, P. Penna, and P. Widmayer. Online train disposition: to wait or not to wait? *Electronic Notes in Theoretical Computer Science*, 66(6), 2002.

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

[Car99] M. Carey. Ex ante heuristic measures of schedule reliability. *Transportation Research*, 53B(3):473–494, 1999.

[EF02] O. Engelhardt-Funke. *Stochastische Modellierung und Simulation von Verspätungen in Verkehrsnetzen für die Anwendung der Fahrplanoptimierung*. PhD thesis, Univerität Clausthal, 2002.

[EFK01] O. Engelhardt-Funke and M. Kolonko. Simulating delays for realistic timetable-optimization. In *Operations Research Proceedings 2001*, pages 9–15. Springer, 2001.

[Elm77] S.E. Elmaghraby. *Activity Networks*. Wiley Interscience Publication, 1977.

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

[GHL05] L. De Giovanni, G. Heilporn, and M. Labbé. Optimization models for the delay management problem in public transportation. Technical report, Paper presented on ORP3, Valencia, 2005.

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

[GJPW04] M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. On-line delay management on a single train line. Technical report, ETH Zürich, 2004.

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

[GS02] A. Ginkel and A. Schöbel. The bicriterial delay management problem. Technical report, Universität Kaiserslautern, 2002.

[Kli00] N. Kliewer. Mathematische Optimierung zur Unterstützung kundenorientierter Disposition im Schienenverkehr. Master's thesis, Universität Paderborn, 2000.

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

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[Pee03] L. Peeters. *Cyclic Railway Timetabling Optimization*. PhD thesis, ERIM, Rotterdam School of Management, 2003.

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

[SBK01] L. Suhl, C. Biederbick, and N. Kliewer. Design of customer-oriented dispatching support for railways. In S. Voß and J. Daduna, editors, *Computer-Aided Transit Scheduling*, volume 505 of *Lecture Notes in Economics and Mathematical systems*, pages 365–386. Springer, 2001.

[Sch01a] S. Scholl. Anschlusssicherung bei Verspätungen im ÖPNV. Master's thesis, Universität Kaiserslautern, 2001.

[Sch01b] A. Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

[Sch06] A. Schöbel. *Customer-oriented optimization in public transportation*. Applied Optimization. Springer, New York, 2006. to appear.

[SM01] L. Suhl and T. Mellouli. Managing and preventing delays in railway traffic by simulation and optimization. In *Mathematical Methods on Optimization in Transportation Systems*, pages 3–16. Kluwer, 2001.

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

[SvdB01] B. De Schutter and T. van den Boom. Model predictive control for railway networks. In *Proceedings of the 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 105–110, Como, Italy, 2001.

[vE01] R.J. van Egmond. An algebraic approach for scheduling train movements. In *Proceedings of the 8th international conference on Computer-Aided Transit Scheduling, Berlin, 2000*, 2001.