Rules of Thumb – Practical Online-Strategies for Delay Management

R. Bauer, A. Schöbel

Nr. 2011-3
Abstract. The delay management problem asks how to react to exogenous delays in public railway traffic such that the overall passenger delay is minimized. Such source delays occur in the operational business of public transit and easily make the scheduled timetable infeasible. The delay management problem is further complicated by its online nature. Source delays are not known in advance, hence decisions have to be taken without exactly knowing the future. This work focuses on online delay management.

We enhance established offline models and gain a generic model that is able to cover complex realistic memoryless delay scenarios. We introduce and experimentally evaluate online strategies for delay management that are practical, easily applicable, and robust. The most promising approach is based on simulation and a learning strategy. Finally, by analyzing the solutions found, we gain interesting new insights in the structure of good delay management strategies for real-world railway data.

1 Introduction

The delay management problem asks how to react to exogenous delays (source delays) in public railway traffic such that the overall passenger delay is minimized. These source delays occur in the operational business of public transit and usually make the scheduled timetable infeasible. Many operational constraints have to be taken into account when updating the scheduled timetable to a disposition timetable.

The two main aspects treated in literature are as follows: Firstly, passenger trips often require changing from one train to another. Given a delayed feeding train, a wait-depart decision settles the question if a follow-up train should wait in order to enable changing activities. Secondly, the limited capacity of the track system complicates the creation of a good disposition timetable. Headway constraints model this limited capacity. Every time two trains simultaneously compete for the same part of the train system, it has to be decided which train may go first. What has been neglected in most publications so far is the online nature of the problem. Source delays are usually not known in advance, hence decisions have to be taken without exactly knowing the future.

CONTRIBUTION. This work focuses on online delay management. We enhance both offline models given in [1] for the online case: The uncapacitated model that concentrates on wait-depart decisions and the more general model that additionally considers the limited capacity of the track system. We gain a generic model that is able to cover complex realistic memoryless delay scenarios as well as standard academic delay scenarios that require knowing the past.

In particular, we introduce and experimentally evaluate online strategies for delay management. Besides the quality of the online solution, our aim also is to find strategies that are practical in the sense that they are easily applicable and robust. We propose for the first time a learning-strategy for online delay management. It does not need complete information on the state of the entire system and is hence simple and robust, but nevertheless turns out to be superior to other heuristics proposed in the literature and even to ILP-approaches. We compare our results to tight a-posteriori bounds given by an optimal offline solution. Finally, by analyzing the solutions found, we gain interesting new insights in the structure of good delay management strategies for real-world railway data.

RELATED WORK. There exist various models and solution approaches for delay management, mainly treating its offline version. If capacities are neglected the question is to decide which trains should wait for delayed feeder trains and which trains better depart on time. A first integer programming formulation

* Partially supported by the DFG (projects WAG54/16-1 and SCH 1140/3-1).
for this problem has been given in [2] and has been further developed in [3] and [4]; see also [5] for an overview on various models. The complexity of the problem has been investigated in [6, 7] where it turns out that the problem is NP-hard even in very special cases.

Further publications about delay management include a model in the context of max-plus-algebra [8, 9], a formulation as discrete time-cost tradeoff problem [10] and simulation approaches [11, 12]. Recently, also the limited capacity of the track system has been taken into account, see [13] for modeling issues and [1, 14] for an extensive analysis of the resulting integer program and heuristic approaches solving the capacitated delay management problem. A model which includes the routing of the passengers can be found in [15].

An online version of the problem has been studied in [16, 17], where its relation to job-shop scheduling is pointed out and Graham’s algorithm is studied. In [18], it was shown that the online version of the delay management problem is PSPACE-hard. In [11], an online version of delay management was studied in which priority-based strategies were compared with the according optimal offline solution and with a solution resulting from an optimal recomputation in each step. The optimal offline solution and the optimal recomputation are gained by an ILP-formulation. The underlying model is similar to the one used in this work. However, it differs in some aspects, such as the objective function and the existence of headway constraints.

**Overview.** In Section 2 we formally state the problem. Section 3 introduces the considered delay management strategies. Experimental results are given in Section 4. The work ends with a conclusion in Section 5.

## 2 Problem Statement

Given two vectors \( a, b \in \mathbb{R}^k \) for some \( k \) we write \( a \leq b \) if \( a_i \leq b_i \) for each \( i = 1, \ldots, k \). We use the convention \( \infty + \infty = \infty \).

**Model of the Railway System.** An *event-activity* network is a directed graph \( \mathcal{N} = (\mathcal{E}, \mathcal{A}) \) where \( \mathcal{E} = \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}} \) is decomposed into arrival and departure events \( \mathcal{E}_{\text{arr}} \) and \( \mathcal{E}_{\text{dep}} \). Each node in \( \mathcal{E} \) corresponds to a specific train arriving or departing at a specific time in or from a specific station. An edge in \( \mathcal{A} \) is called an *activity*. The set of activities partitions into \( \mathcal{A} = \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{head}} \) with each activity \( a \in \mathcal{A} \) having a minimal duration \( L_a > 0 \). Structure and meaning of the activities is as follows:

- A *driving activity* \( a \in \mathcal{A}_{\text{drive}} \subseteq \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{arr}} \) represents a train driving between two consecutive stations.
- A *waiting activity* \( a \in \mathcal{A}_{\text{wait}} \subseteq \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}} \) corresponds to the time period in which a train is waiting in a station to let passengers on or off.
- A *changing activity* \( a \in \mathcal{A}_{\text{change}} \subseteq \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}} \) corresponds to the transfer of passengers from one train to another (by foot, within a station).
- A *headway activity* \( a \in \mathcal{A}_{\text{head}} \subseteq \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{dep}} \) models the limited capacity of the track system. This can either be two trains driving on the same track into the same direction or two trains driving into opposite directions on a single-way track. The duration \( L_{(i,j)} \) of a headway activity \((i,j)\) means that the departure \( j \) must take place at least \( L_{(i,j)} \) minutes after the departure \( i \) (if \( j \) actually takes place after \( i \)).

For each changing activity \( \mathcal{A}_{\text{change}} \) the delay management problem asks to decide if we leave it in the network (i.e., if the connection is maintained) or if it is deleted (if the connection is not maintained). For each activity \((i,j)\) in \( \mathcal{A}_{\text{head}} \), it is \((j,i)\) in \( \mathcal{A}_{\text{head}} \). If \((i,j)\) is respected this means that \( i \) happens before \( j \). The delay management problem decides which of these two constraints is respected and which is dropped. There is no such decision to be made for \( \mathcal{A}_{\text{drive}} \) and \( \mathcal{A}_{\text{wait}} \). Hence we abbreviate \( \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \) by \( \mathcal{A}_{\text{train}} \).

**Definition (timetable).** A *timetable* for \( \mathcal{N} \) is a vector \( x \in \mathbb{N}^{|\mathcal{E}|} \) which assigns a time \( x_i \in \mathbb{N} \) to each event \( i \in \mathcal{E} \) such that

\[
\begin{align*}
    x_j - x_i & \geq L_{(i,j)} & &\text{for each activity } (i,j) \in \mathcal{A}_{\text{train}} \\
x_j - x_i & \geq L_{(i,j)} \or x_i - x_j & \geq L_{(j,i)} & &\text{for each activity } (i,j) \in \mathcal{A}_{\text{head}}
\end{align*}
\]
Throughout this work we assume that an event-activity network \( \mathcal{N} = (\mathcal{E}, \mathcal{A}) \) is given. The network will always be annotated with the following information: For each event \( i \in \mathcal{E}_{\text{arr}} \) the number of passengers \( w_i \) getting off at event \( i \) and arriving there at their final destination, and for all \( a \in \mathcal{A}_{\text{change}} \) the number of passengers \( w_a \) who want to use activity \( a \). We always assume \( w_a > 0 \) for each \( a \in \mathcal{A}_{\text{change}} \). Further, a number \( T \in \mathbb{N} \) is given. We interpret \( T \) as a common time-period for all lines of the underlying railway system and use it as a penalty for not maintaining a changing activity. Finally, we are given a timetable \( \pi \) for \( \mathcal{N} \) which we call the scheduled timetable. This timetable maintains every changing activity of \( \mathcal{N} \), that is, \( \pi_i + L_{(i,j)} \leq \pi_j \) for all \( (i, j) \in \mathcal{A}_{\text{change}} \). Note that the graph \( (\mathcal{E}, \mathcal{A}_{\text{\neg \text{head}}}) \) is acyclic as otherwise \( \pi \) would not exist. More detailed explanations on this model can be found in [14] and [1], a small example of an event-activity network is given in Figure 1.

**Fig. 1**: Example of an event-activity network. Solid edges represent driving and waiting activities, dotted edges represent changing activities and dashed edges represent headway activities.

**MODELS FOR OFFLINE DELAY MANAGEMENT.** A (source) delay state \( d \) reflects the current knowledge and expectations on exogenous delays in the underlying railway system. Such delays are called source delays. Formally, \( d \) is a mapping from \( \mathcal{E} \cup \mathcal{A}_{\text{train}} \) to \( \mathbb{N} \) meaning that there is a source delay of \( d(w) \) at activity/event \( w \). We write \( d_{(i,j)} \) and \( d_i \) for \( d(i, j) \) and \( d(i) \), respectively. Whenever we become aware of altered source delays, the scheduled timetable \( \pi \) has to be updated to a so-called disposition timetable \( x \in \mathbb{N}^{\mathcal{E}} \).

**Definition (Disposition timetable).** Given a delay state \( d \), a disposition timetable \( x \in \mathbb{N}^{\mathcal{E}} \) for \( d \) is a timetable for \( \mathcal{N} \) such that

\[
x_i \geq \pi_i + d_i \quad \text{for each event } i \in \mathcal{E} \tag{3}
\]

\[
x_j \geq x_i + d_{(i,j)} + L_{(i,j)} \quad \text{for each activity } (i, j) \in \mathcal{A}_{\text{train}}. \tag{4}
\]

Given a disposition timetable \( x \) and a changing activity \( (i, j) \) we write \( z^x_{(i,j)} = 0 \) if \( x_j \geq x_i + L_{(i,j)} \) and \( z^x_{(i,j)} = 1 \) otherwise. That means \( z^x_{(i,j)} \) equals 0 if and only if the changing activity \( (i, j) \) is maintained by the disposition timetable \( x \). The overall delay \( f(x) \) of a disposition timetable \( x \) is

\[
f(x) = \sum_{i \in \mathcal{E}_{\text{arr}}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z^x_{(i,j)} w_a T.
\]
approximating the overall delay of passenger arrivals according to the timetable \( x \). Offline delay management assumes that all source delays are known in advance.

**Problem 1 (Capacitated Offline Delay-Management).** Given a timetable \( \pi \) and a delay state \( d \), find a disposition timetable \( x \) for \( d \) of minimal overall delay \( f(x) \).

The uncapacitated case does not take the limited capacity of the track system into account.

**Problem 2 (Uncapacitated Offline Delay-Management).** Given a timetable \( \pi \) and a delay state \( d \), find a disposition timetable \( x \) for \( d \) on \( N' = (E,A\setminus A_{\text{head}}) \) of minimal overall delay \( f(x) \).

**MODELS FOR ONLINE DELAY MANAGEMENT.** In the online case, source delays are not completely known in advance and expectations may change over time. We model this as a process of repeatedly updated knowledge and expectations on future delays. We are led by the following considerations: We want to be able to model completely unexpected delays (like accidents) that get known just at the time the respective event is scheduled. We want to be able to model changing expectations (like changing plans for working sites). The knowledge and expectation on delays are given by a finite sequence \( \sigma = (d^1,t^1), (d^2,t^2), \ldots, (d^m,t^m) \) where \( d^k \) is a delay state and \( t^k \in \mathbb{N} \) the time at which \( d^k \) becomes active. Between time \( t^k \) and \( t^{k+1} \) everything happens as expected in state \( d^k \).

From a more technical point of view, we have the request to be able to express realistic, memoryless models that work similar to Markov-chains as well as distributions applied in academics for experiments. The following definitions of feasible next delay state and online strategy for delay management give the minimal technical restrictions that are necessary to fulfill the above desiderata.

Assume that at time \( t^1 \), the current delay state is \( d^1 \) and the disposition timetable is \( x^1 \). At a point \( t^2 > t^1 \) in time, the delay state changes to a new state \( d^2 \). We assume that all events and delays that were scheduled between \( t^1 \) and \( t^2 \) happened as planned in \( x^1 \). Hence, \( d^1 \) and \( d^2 \) are equal for all activities \( (i,j) \) with \( x^1_{ij} < t^2 \). A delay state \( d^2 \) that guarantees this property is called a feasible next delay state.

**Definition (Feasible next delay state).** Given delay states \( d^1 \) and \( d^2 \), a timetable \( x^1 \) and a point in time \( t^2 \in \mathbb{N} \), we call \( (d^2,t^2) \) feasible for \( (x^1,d^1) \) if \( d^1_{(i,j)} = d^2_{(i,j)} \) for each activity \( (i,j) \) with \( x^1_{ij} < t^2 \) and \( d^1_j = d^2_j \) for each event with \( x^1_{ij} < t^2 \).

At time \( t^2 \), it may turn out that the timetable \( x^1 \) is not feasible any more. An online strategy (for delay management) adapts a timetable to fit the new delay state. We assume that the re-scheduling is done at time \( t^2 \) and all events that were scheduled in \( x^1 \) before time \( t^2 \) already happened as planned. Hence, only events scheduled (according to \( x^1 \)) after time \( t^2 \) can be rescheduled.

**Definition (Online Strategy for Delay Management).** Given are a timetable \( x^1 \), a time \( t^2 \in \mathbb{N} \) and delay states \( d^1, d^2 \) such that \( (d^2,t^2) \) are feasible for \( (x^1,d^1) \). An online-strategy for delay management is an algorithm \( S \) that computes a disposition timetable \( x^2 = S(x^1,d^2,t^2) \) for \( d^2 \) such that

\[
\begin{align*}
\bullet & \ x^2_i = x^1_i \text{ for each } i \in E \text{ with } x^1_i < t^2 & (5) \\
\bullet & \text{from } x^1_i \geq t^2 \text{ follows } x^2_i \geq t^2 . & (6)
\end{align*}
\]

Accordingly, we express the current state of the system through the triple \((x,d,t)\) of current disposition timetable \( x \), delay state \( d \) and a time \( t \) representing the start of \( d \). Then, next delay state and time \((d',t')\) are randomly chosen by a delay generator.

**Definition (Delay Generator).** Let \( D \) be a black-box routine whose input may consist of current disposition timetable \( x \), delay state \( d \), time \( t \), additional random values and values computed in former executions of \( D \). We call \( D \) a delay generator if \( (d',t') := D(x,d,t) \) is a feasible next delay state for input \((x,d,t)\) and any additional input.
Starting with \((x, d, t) = (\pi, 0, 0)\) online delay management first obtains a tuple \((d', t')\) from \(\mathcal{D}\). Then, a new disposition timetable \(x'\) is computed by a delay management strategy \(\mathcal{S}\). The process iteratively repeats until the considered time horizon ends (i.e., it stops, when the roll-out time is reached). The objective function (the delay of all passengers) is computed out of the final disposition timetable.

**Definition (Delay Management Process).** We are given a delay management strategy \(\mathcal{S}\) and a delay generator \(\mathcal{D}\). The delay management process \(\mathcal{M}\) with respect to \(\mathcal{S}\) and \(\mathcal{D}\) is the process constructed as follows:

Starting with \(x^0 = \pi, d^0 \equiv 0\) and \(t^0 = 0\), we iteratively obtain \((d^{i+1}, t^{i+1}) := \mathcal{D}(x^i, d^i, t^i)\) and afterwards compute \(x^{i+1} = \mathcal{S}(x^i, d^{i+1}, t^{i+1})\). The process ends when \(t^{m+1} = \infty\) (and there with the convention \(x^{m+1} = x^m\) and \(d^{m+1} = d^m\)). Let \(\sigma = (x^1, d^1, t^1), (x^2, d^2, t^2), \ldots, (x^m, d^m, t^m)\) be a realization of \(\mathcal{M}\). The overall delay of \(\sigma\) is \(\int(x^m)\).

The process is depicted in Figure 2. At a first glance it might look unnecessary that the delay generator also incorporates the current disposition timetable \(x\) as an input. The reason is the following: In order to have the next delay state feasible, the delay generator has to assure that events that already happened do not change. In order to know which events already happened it is necessary to know the current disposition timetable \(x\). Online Delay Management aims at finding good average case strategies.

**Problem 3 (Capacitated Online Delay Management).** Given a delay generator \(\mathcal{D}\), find an online strategy \(\mathcal{S}\), such that the expected overall delay of the delay process \(\mathcal{M}\) of \(\mathcal{S}\) and \(\mathcal{D}\) is minimal.

Again, the uncapacitated case does not take the limited capacity of the track system into account.

**Problem 4 (Uncapacitated Online Delay Management).** Uncapacitated Online Delay Management is Capacitated Online Delay Management on \(\mathcal{N}' = (\mathcal{E}, \mathcal{A} \setminus \mathcal{A}_{\text{head}})\).

\[ \begin{align*}
\text{timetable } x^0 &= \pi \\
\text{timetable } x^1 &\xrightarrow{\mathcal{S}(x^0, d^0, t^0)} \\
\text{timetable } x^2 &\xrightarrow{\mathcal{S}(x^1, d^1, t^1)} \xrightarrow{\mathcal{D}(x^1, d^1, t^1)} \\
\text{timetable } x^3 &= x^m \\
&\xrightarrow{f(x^m)} \\
\text{delay state } d^0 &= 0 \\
\text{time } t^0 &= 0 \\
\text{delay state } d^1 &\rightarrow \\
\text{time } t^1 &= \\
\text{delay state } d^2 &\rightarrow \\
\text{time } t^2 &= \\
\text{delay state } d^3 &\rightarrow \\
\text{time } t^3 &= \\
\end{align*} \]

**Fig. 2: Illustration of the delay management process for a delay management strategy \(\mathcal{S}\).**

**A DELAY GENERATOR.** Within our experiments we use the delay generator DDA (delays determined in advance) which is described in the following. We assume that delays do not depend on the delay management strategy and a final delay state of \(d^\text{final}\) is chosen before the actual delay management process starts. Note that we still start the process with state \((x^0 = \Pi, d^0 = 0, t^0 = 0)\).

Delays get known over time, directly when they are scheduled to happen, i.e., to compute \((d^{k+1}, t^{k+1}) := \mathcal{D}(x^k, d^k, t^k)\) we search for the next (according to \(x^k\)) set of events or activities that are source delayed (according to \(d^\text{final}\)). More formally, let

\[- i \in \mathcal{E}\] be such that \(d^k_i = 0, d^\text{final}_i > 0\) and \(x^k_i\) is minimal, and

\[- (j, w) \in \mathcal{A}\] be such that \(d^k_{(j, w)} = 0, d^\text{final}_{(j, w)} > 0\) and \(x^k_j\) is minimal

We set \(t^{k+1} = \min\{x^k_i, x^k_j\}\). Given arbitrary \(a \in \mathcal{E}, (v, w) \in \mathcal{A}_{\text{train}}\), the next delay state \(d^{k+1}\) is given by

\[
d_a^{k+1} = \begin{cases} 
  d^\text{final}_a &, x_a^k \leq t^{k+1} \text{ or } d_a^k > 0 \\
  0 &, \text{otherwise}
\end{cases}
\]

\[
d_{(v, w)}^{k+1} = \begin{cases} 
  d^\text{final}_{(v, w)} &, x_{(v, w)}^k \leq t^{k+1} \text{ or } d_{(v, w)}^k > 0 \\
  0 &, \text{otherwise}
\end{cases}
\]
We finish with time $t^{k+1} = \infty$ when $d^k = d^\text{final}$. As the delay management process gets deterministic as soon as $d^\text{final}$ is fixed, we call $d^\text{final}$ an instance of generator DDA.

3 Delay Management Strategies

AN A-PERIODIC BOUND AND ILP-APPROACHES. In [1] an exact integer programming formulation for the offline problem is given. The formulation is as follows.

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

such that

$$
\begin{align*}
& x_i \geq \pi_i + d_i & i \in \mathcal{E} \\
& x_i - x_j \geq L(i,j) + d_{i,j} & (i,j) \in \mathcal{A}_{\text{train}} \\
& Mz_{(i,j)} + x_j \geq L(i,j) + x_i & (i,j) \in \mathcal{A}_{\text{change}} \\
& Mg_{ij} + x_j \geq L(i,j) + x_i & (i,j) \in \mathcal{A}_{\text{head}} \\
& g_{ij} + g_{ji} = 1 & (i,j) \in \mathcal{A}_{\text{head}} \\
& x_i \in \mathbb{Z}^+ & i \in \mathcal{E} \\
& z_a \in \{0, 1\} & a \in \mathcal{A}_{\text{change}} \\
& g_{ij} \in \{0, 1\} & (i,j) \in \mathcal{A}_{\text{head}} \\
\end{align*}
$$

where $M$ is a number ‘big enough’. Detailed explanations on the ILP including a discussion on the size of $M$ can be found in the original work.

When working with delay generator DDA, all source-delays are determined by $d^\text{final}$ before the delay management process starts. Consider an optimal solution to the offline problem for $d^\text{final}$. This solution obviously gives an a-posteriori bound on the corresponding online problem for that instance. The following lemma shows that this bound is tight.

Lemma 1. Let $d^\text{final}$ be an instance of delay generator DDA. Let $x^{\text{opt}}$ be an optimal solution to the offline problem for $d^\text{final}$. Then, there is an online strategy $\mathcal{S}$ that generates (on instance $d^\text{final}$) a delay management process $\sigma = (x^1, d^0, t^0), (x^2, d^1, t^1), \ldots, (x^m, d^{m-1}, t^{m-1})$ such that $f(x^{\text{opt}}) = f(x^m)$.

Proof (of Lemma 1). Given $d^\text{final}$, the following online strategy does the desired. Let $t_1 = \min\{\pi_i | i \in \mathcal{E} \land d_i > 0 \text{ or } (i,j) \in \mathcal{A} \land d_{i,j} > 0\}$ and let $x^0 := \pi$. For $k \geq 1$, we set $S(x^k, d^{k+1}, t^{k+1})$ to be a timetable $x'$ which is defined by $x'_i = \pi_i$ if $\pi_i < t_1$ and $x'_i = x_i^{\text{opt}}$ otherwise.

Obviously, $S$ is an online strategy for $k > 0$. Because of $x_i^{\text{opt}} \geq \pi_i$ for any $i \in \mathcal{E}$, we have $w_i (x_i^{\text{opt}} - \pi_i) \geq w_i (x_i - \pi_i)$. Further, it is $z_i^{\text{opt}} = 0$ if $\pi_i < t_1$. (This holds in the case that $\pi_i < t_1$ since then both events are not delayed, and in the case that $\pi_i \geq t_1$ since the transfer can take place if only the departing train has a delay). Accordingly, $z_i^{\text{opt}} = z_i^{\text{opt}}$ if $\pi_i, \pi_j \geq t_1$.

Consequently we have $z_i^{\text{opt}} w_i T \leq z_i^{\text{opt}} w_i T$. Summarizing, it is $f(x') \leq f(x^{\text{opt}})$. To prove $f(x^{\text{opt}}) \geq f(x^m)$, it remains to show that $x^m$ is a disposition timetable for each $d^k$ which can easily be done by checking Equations 2, 3 and 4. It is $f(x^{\text{opt}}) \leq f(x^m)$ as $x^m$ must be a disposition timetable for $d^m$. \qed

By fixing past events and iteratively recomputing the exact offline-solution we can use the ILP as an online strategy:

Strategy (OnlineILP). Given $(x^k, d^{k+1}, t^{k+1})$, compute $x^{k+1} = S(x^k, d^{k+1}, t^{k+1})$ by setting $x^{k+1} = x$ for a feasible solution $(x, z, g)$ of

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

such that (10)-(17) and such that

$$
\begin{align*}
x_i &= x_i^k & i \in \mathcal{E} \text{ and } x_i^k < t^{k+1} \\
x_i &\geq t^{k+1} & i \in \mathcal{E} \text{ and } x_i^k \geq t^{k+1}
\end{align*}
$$
**AD-HOC RE-SCHEDULING.** This online strategy iteratively re-schedules the timetable event-by-event. Simple heuristic functions are used to decide time and choice of the next event to schedule. The approach uses only local information when scheduling an event. The resulting disposition timetable strongly depends on the choice of the heuristic functions. Without considering headway constraints the strategy is straightforward: First, order the events that may be influenced by the given delay topologically. Then proceed the events in this order and let each event start as early as possible with respect to the applied heuristic. If headways are regarded, we further refine the topological order and have to obey more restrictions. The computation of the next disposition timetable $x^{k+1} = S(x^k, d^{k+1}, t^{k+1})$ then works as follows:

For each event $i$, the time $x^{k+1}_i$ is initialized to $x^k_i$ if $x^k_i < t^{k+1}$. Otherwise we initialize $x^{k+1}_i$ with $\infty$. During the run of the algorithm, $x^{k+1}_i = \infty$ indicates that event $i$ has not yet been scheduled. For each event $i$ we can compute a wish time $(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i)$ for the event to happen. This wish depends on the choice of the already re-scheduled events. It consists of three parts.

$$\text{time}(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i) := \max\{\text{time}_{\text{top}}(i, x^{k+1}_i),$$
$$\text{time}_{\text{earliest}}(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i),$$
$$\text{time}_{\text{dm}}(i, x^{k+1}_i, d^{k+1}_i)\}$$

The function $\text{time}_{\text{earliest}}(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i)$ assures that all technical restrictions are respected and hence makes sure that we gain a (feasible) disposition timetable. The function $\text{time}_{\text{dm}}(i, x^{k+1}_i, d^{k+1}_i)$ realizes the applied heuristics while $\text{time}_{\text{top}}(i, x^{k+1}_i)$ influences the order in which events are scheduled. All three functions will be described later.

Now, while there is an unscheduled event, we pick an arbitrary unscheduled event $i$ with minimum time $(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i)$ and schedule it to happen at that time. Note that scheduling an event $i$ may alter the computation of the next disposition timetable for other events $j$. The pseudocode is given as Algorithm 1. Each computation step needs only local information. Hence, only small changes are required such that the approach can be applied locally by the trains drivers or by the local disponents at single train stations.

**Algorithm 1: Ad-Hoc Re-Scheduling**

```
input : time $x^{k+1} \in \mathbb{N}$, old disposition timetable $x^k$, new delay state $d^{k+1}$, function time($\cdot, \cdot, \cdot$)
output : new disposition timetable $x^{k+1}$
1 for $i \in \mathcal{E}$ do
2    $x^{k+1}_i \leftarrow \infty$;
3    if $x^k_i < t^{k+1}_i$ then $x^{k+1}_i \leftarrow x^k_i$
4 while there is an event $i$ with $x^{k+1}_i = \infty$ do
5    $i \leftarrow$ choose an arbitrary element in $\{i \in \mathcal{E} \mid x^{k+1}_i = \infty\}$ with minimal time $(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i)$;
6    $x^{k+1}_i \leftarrow \text{time}(i, x^{k+1}_i, d^{k+1}_i, t^{k+1}_i)$
```

**COMMON PARTS OF ALL HEURISTICS.** While the function $\text{time}_{\text{dm}}$ contains the individual part of each heuristic, all applied strategies share the same choice of $\text{time}_{\text{earliest}}$ and $\text{time}_{\text{top}}$.

The graph $\mathcal{N} = (\mathcal{E}, \mathcal{A} \setminus \mathcal{A}_{\text{head}})$ is acyclic. It turned out that the quality of Ad-Hoc Re-Scheduling significantly increases when we perform the scheduling of the events in topological order of $\mathcal{N}$. The reason is, that we otherwise loose information when computing $\text{time}_{\text{dm}}$. Given an event $i$, a successor $j$ of $i$ in $\mathcal{N}$ and the resulting disposition timetable $x$, that does not necessarily mean that $x_i \leq x_j$ but that we determine the value of $x_j$ before we determine the value of $x_i$. It further is possible that we determine the value of $x_i$ before we determine the value of $x_j$ even if $x_i > x_j$. The function

$$\text{time}_{\text{top}}(i, x) := \begin{cases} 
\infty & \text{there is an } (j, i) \in \mathcal{A}_{\text{change}} \text{ with } x_j = \infty \\
0 & \text{otherwise}
\end{cases}$$

7
ensures that the topological order is respected for changing activities. The task of function \( \text{time}_{\text{earliest}}(i, x, d, t) \) is to compute the earliest point in time at which event \( i \) can take place with respect to the original timetable \( \pi \), the preceding event on the same line, the current time \( t \), the headway constraints of all scheduled events and all known source delays. This also ensures that the topological order is guaranteed for waiting and driving activities. The value

\[
\text{earliestNoHead}(i, x, d, t) := \max \left\{ \{ t, \pi_i + d_i \} \cup \{ x_j + L(j,i) + d(j,i) \mid (j, i) \in \mathcal{A}_{\text{train}} \} \right\}
\]

already respects all requirements but the headways. The function \( \text{nextSlot} \) considers the headway constraints by giving the earliest point in time \( at \ or \ after \) the time \( t \) at which event \( i \) can take place without violating a headway constraint of an already scheduled event, i.e., an event \( j \) with \( x_j < \infty \).

\[
\text{nextSlot}(i, x, t) := \min \{ t \geq t | t \not\in [x_j, x_j + L(j,i)] \text{ for } (j, i) \in \mathcal{A}_{\text{head}}, x_j < \infty \} \quad x_j \not\in [t, t + L(i,j)] \text{ for } (i, j) \in \mathcal{A}_{\text{head}}, x_j < \infty \}
\]

Note that this function may be suboptimal in case of \( L(i,j) = 0 \), but this can easily be handled as a special case. The function \( \text{time}_{\text{earliest}} \) combines \( \text{nextSlot} \) and \( \text{earliestNoHead} \)

\[
\text{time}_{\text{earliest}}(i, x, d, t) := \text{nextSlot}(i, x, \text{earliestNoHead}(i, x, d, t))
\]

and hence satisfies all technical requirements.

**Uniform-Heuristics for Ad-Hoc Re-Scheduling.** This class of Ad-hoc Re-Scheduling uses the same decision strategy for all events and does not depend on preprocessed information. The function \( \text{time}_{\text{dm}} \) determines whether a train should wait for a feeder train. In the following, we use the indicator function \( 1_A(y) \) given by \( 1_A(y) = y \) if \( y \in A \) and 0 otherwise. and the convention \( \min \{0\} = 0 \). Given a parameter \( \ell_i \), we define \( \text{time}_{\text{dm}} \) as

\[
\text{time}_{\text{dm}}(i, x, d, t) := \max \left\{ 1_{[0, \ell_i]}(\text{nextSlot}(i, x, x_j + L(j,i) + d(j,i))) \mid (j, i) \in \mathcal{A}_{\text{change}} \right\}.
\]

The value of \( \ell_i \) steers how long event \( i \) waits for feeder trains. We apply the following choices.

- **Always wait.** Formally: \( \ell_i = \infty \)
- **Never wait.** Formally: \( \ell_i = 0 \)
- **Wait if enough slack on next activity.**
  - Formally: \( \ell_i = \min \{ \pi_w - L(j,w) - d(j,w) \mid (j, w) \in \mathcal{A}_{\text{train}} \} \)
- **Wait \( y \) minutes starting from earliest possible departure.**
  - Formally: \( \ell_i = \text{time}_{\text{earliest}}(i, x, d, t) + y \)
- **Wait \( y \) minutes starting from original timetable.** Formally: \( \ell_i = \pi_i + y \)

The strategies **always wait, never wait, and wait \( y \) minutes starting from original timetable** have been used before in [11], but without respecting headway constraints.

**Learning Non-Uniform Heuristics from Simulation.** In the non-uniform case, we pre-process some data that is used as a parameter when computing priorities. For each changing activity \( (i, j) \) we are given a value \( \ell_{(i,j)} \in \mathbb{N} \) meaning that event \( j \) should wait at most \( \ell_{(i,j)} \) minutes starting from the scheduled timetable in order to maintain \( (i, j) \). Accordingly, we have

\[
\text{time}_{\text{dm}}(i, x, d) = \max \left\{ 1_{[0, \ell_{(i,j)}]}(x_j + L(j,i) + d(j,i)) \right\}.
\]

To learn the values of \( \ell_{(i,j)} \) we first generate a number of random delay states. In case of model DDA we use the final delay states \( d_{\text{final}} \). Otherwise it is reasonable to obtain final delay states by applying an arbitrary delay management strategy. Afterwards we exactly solve the offline problem on these instances using the ILP. We obtain a sequence \( (x^1, d^1), \ldots, (x^n, d^n) \) where disposition timetable \( x^i \) is an optimal
solution of the random instance $d^i$. In order to obtain $\ell_{(i,j)}$ for a changing activity $(i, j)$, we compute the multisets

$$\text{yes}_{(i,j)} := \left\{ x^k_j + L_{(i,j)} - \pi_j^i \mid x^k_j \geq x^k_i + L_{(i,j)}, k \in 1, \ldots, m \right\}$$

$$\text{no}_{(i,j)} := \left\{ x^k_j + L_{(i,j)} - \pi_j^i \mid x^k_j < x^k_i + L_{(i,j)}, k \in 1, \ldots, m \right\}.$$ 

These sets contain, for each instance, the value how long event $j$ would have had to wait in order to maintain activity $(i, j)$. The simulated instances $x^1, \ldots, x^m$ are split such that $\text{yes}_{(i,j)}$ represents all instances in which $(i, j)$ actually has been maintained and such that $\text{no}_{(i,j)}$ represents the remaining instances. We let $\ell_{(i,j)}$ be an arbitrary value that separates $\text{yes}_{(i,j)}$ and $\text{no}_{(i,j)}$ optimally. That is a value $\ell_{(i,j)}$ for which

$$p(\ell_{(i,j)}) := \left| \{ v \in \text{yes}_{(i,j)} \mid v > \ell_{(i,j)} \} \right| + \left| \{ v \in \text{no}_{(i,j)} \mid v < \ell_{(i,j)} \} \right|$$

is minimal. Experimentally, we made the observation that it usually is possible to split both sets very well, often with $p(\ell_{(i,j)}) = 0$. Further, when considering the quality of the online-phase, the actual choice of $\ell_{(i,j)}$ out of all optimal values only mattered for very small simulation numbers $m$.

4 Experiments

In this section, we present an experimental evaluation of the algorithms described above. Unless stated otherwise experiments are performed on a random sample of size 100 and the preprocessing of the learning heuristics uses 1000 simulation runs.

All experiments are reported as box-and-whiskers plots: There is one entry on the $x$-axis for every online delay management strategy applied. The $y$-axis always gives the relative quality of the solutions compared to the respective optimal solutions (i.e., the objective value of the online strategy divided by the objective value of the tight lower bound described in Section 3). Bottom and top of the box give the lower and upper quartiles, the band inside a box gives the median and the whiskers give the smallest and highest observations without outliers. We classify any observation outside 1.5 interquartile range of the lower quartile or outside 1.5 interquartile range of the upper quartile as outlier and depict it as a circle.

**Instances.** We work on two different datasets. LinTim is the medium sized real-world instance that is part of the LinTim-package. See [19] for further details. Until now, the dataset does not incorporate headways. The instance HARZ is a real-world dataset representing the network of Deutsche Bahn in the Harz region. The instance incorporates headways. Further information on that data can be found in [14]. For both datasets, the roll-out time gives the length of the considered time horizon.

We use the delay generator DDA. We generate an instance $d^{\text{final}}$ by choosing a given number of activities uniformly at random. Each of them is assigned a random delay, uniformly distributed in the interval [1min, 3min] (scenario WEAK), [3min, 15min] (scenario MEDIUM) or [15min, 18min] (scenario STRONG). We use the scheme NETWORK-NAME/ROLL-OUT-TIME/NUMBER OF DELAYS/Delay-SCENARIO for identifying particular instances. We add the postfix /NOHead for HARZ-instances with headway constraints removed.

**Small observations made by pretests.** It turned out that the strategies ‘wait $y$ minutes from initial timetable’ and ‘wait $y$ minutes from earliest possible departure’ perform very similar. Hence we only include ‘wait $y$ minutes from initial timetable’ in our experimental study.

When learning the values $\ell_{(i,j)}$ from simulation, there are many possibilities of how to treat changing activities for which no data has been gained by the simulation. Further, even in case there is data available, $\ell_{(i,j)}$ usually can be chosen out of one (or even more intervals). In our experiments the actual choice of these degrees of freedom only mattered for very small simulation numbers.

**Runtime.** Our implementation is written in Java using XPRESS as ILP-Solver and the tests were executed on one core of an AMD Opteron 2218, running SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with Java 1.6. Because of the different focus of the problem we do not report the runtimes in great detail or measure highly accurate. Further, we observed that the applied ILP-solver slightly slowed down with growing
number of simulation runs. Hence, the given numbers shall only give an impression on the required runtime.

The average time needed for solving the ILPs mainly depends on size and structure of the network and the roll-out time. The approximate numbers in seconds are < 1 for LINTim/2H, 3 for LINTim/4H, 5 for LINTim/6H, 10 for LINTim/8H, < 1 for HARZ/2H, 9 for HARZ/4H, 48 for HARZ/6H and 234 for HARZ/8H. The runtime of Ad-Hoc Re-Scheduling is neglectable. On the instances of this study it usually took less than 1 second to solve one offline problem.

![Fig. 3: Experiments on the Harz instances with headways removed.](image)

**Main Experiments.** Our main experiments are given as Figures 4 and 3. We observe that the OnlineILP and the learning heuristics are both performing very well. Both strategies are always better than the best uniform strategy and often are very close to the optimum. Note that in [11], the online strategy based on approximating (ILP) did not perform that well.

All experiments on the learning heuristics have been made using 1000 simulation runs. More simulation runs lead to more accurate values of $\ell_a$ and more activities for which estimations of $\ell_a$ are available. Accordingly, in Figure 5 we see that the quality of the approach strongly depends on the number of simulation runs. Hence, the already good quality of the approach can be further improved by using a longer simulation phase.

There are noticeable, extreme outliers on the weak-instances. The reason is simple: On these instances, the part of the objective function representing delayed arrivals in stations is so small that any wrong wait-decision leads to these extreme values.

We now have a short look at the HARZ/NOHEAD-instances. Comparing the strategies against each other, the situation here is similar to the LinTim-instances. The OnlineILP performs slightly worse on HARZ/NOHEAD and there is a significantly smaller gap between the always-wait and the never-wait strategy. A big difference lies in the absolute approximation-ratios. These are much better for all strategies on HARZ/NOHEAD. We interpret that as follows: There is much less influence of the applied strategy on the objective function and hence, less space for improvement on the HARZ/NOHEAD-instances.

**Robustness Issues.** The preprocessing phase of our learning heuristics makes use of knowledge on the underlying delay distribution. Hence we tested the robustness of the approach with respect to deviations in the delay distribution. To this end, we trained our strategy for the delay scenario 20/MEDIUM and used the following scenarios in the online phase: 10/MEDIUM, 60/MEDIUM, 20/WEAK, 20/STRONG. The results are given in Figure 6. The approach shows to be robust with respect to the realized delays.
Fig. 4: Main experiments on the LinTim-instances. The number of simulation runs for Ad-Hoc Rescheduling is always 1000.
and the solution quality only decreases little. However the scenario STRONG deviates heavy enough from MEDIUM to effect in a noticeable decrease in solution quality. This suggests the following modus operandi for practical application: Perform preprocessing for a ‘standard’ and one or two ‘extreme’ scenarios (incorporating more or stronger delays). In the online phase apply the standard values unless an extreme situation is detected. In case of identifying an extreme situation within the online phase, it is technically unproblematic to immediately switch to the preprocessed data for the according situation.

**Towards Learning of Headways.** The experiments seen so far were based on the uncapacitated model, i.e., aimed at solving Problem 2. However, Ad-hoc Re-Scheduling also computes feasible solutions in the presence of headway constraints. The only realistic dataset incorporating headways we have access to are the HARZ-instances. We did some preliminary tests on these instances. Often all strategies performed very similar and the results were considerably worse than the results obtained for the uncapacitated case. The results of the ‘best’ instance we tested are given in Figure 7.

We did some further diagnostics on the HARZ-instances and came to the following conclusion. When working on the HARZ-instance, the influence of wait-depart decisions on the objective function is rather small even when not considering headway constraints (see Figure 3).

When we additionally consider headway constraints on the HARZ-instances, this little influence is compensated by the impact of the priority decisions (that determine which headway to favor). Ad-Hoc Re-Scheduling handles headway constraints in a first-come first-served manner, which of course is not optimal. In [14] it is shown that this strategy works well for small delays but is not suitable for large delays. In this paper we focus on wait-depart decisions, hence we did not study this issue in detail. However, we checked if the learning heuristics can also be applied to learn priority-decisions.

To that end, we performed simulation runs by optimally solving random instances. For a given solution $x$ and a headway constraint $(i, j)$, we define

$$h^k_i := \max\{\pi_i + d_i, \max\{x^k_w + L_{(w,i)} + d_{(w,i)}|(w,i) \in \mathcal{A}_{\text{train}}\}\}$$

and split the multi-sets

$$\text{prefer}_{i < j} := \left\{ h^k_i - h^k_j | x^k_i < x^k_j, k \in 1, \ldots, m \right\}$$

$$\text{prefer}_{i > j} := \left\{ h^k_i - h^k_j | x^k_i > x^k_j, k \in 1, \ldots, m \right\}$$
as described the previous section. Again, both sets could be separated very well by a single value. This is a strong hint for the applicability of the learning heuristics also on priority decisions.

**Insights in the Structure of Good Solutions.** While the delay management problem allows complex interactions between different regions in the underlying network, our results suggest that the impact of wait-depart decisions is astonishingly local on typical instances. Even more surprising, optimal solutions mostly stick to the following simple decision rule: Maintain changing activity \((i, j)\) if \(j\) has to wait no more than time \(\ell(i, j)\) on event \(i\) (for some value \(\ell(i, j)\) which has to be chosen extremely carefully). As we worked on datasets of different origin we expect that our results can be generalized to a larger class of real-world instances.

**Influence of the Rollout Time.** We also checked if the roll-out time has influence on the performance of the applied strategies. It turned out that this influence is small and the strategies’ performance is quite robust with respect to roll-out time. As a small impact of the roll-out time, good solutions on longer roll-out times tend to have slightly less waiting decisions. This is easily explainable as in this case waiting decisions can propagate further into the future. Some experiments on the roll-out time can be seen in Figure 8.

## 5 Conclusion

This work concentrates on wait-depart decisions for online delay management. We enhance the offline model described in [1] to the online case and only introduce additional restrictions that are required to assure that the delay management process is well-defined. Hence, we gain a generic model that is able to model a wide range of scenarios. Consequently, the considered online strategies only use fundamental information and can be applied to a broad number of different settings.

We state three approaches for solving the problem: An ILP-based approach (that is based on the corresponding offline-ILP), a class of simple ‘rule of thumb’ strategies and a learning heuristics that identifies the structure of good solutions using a simulation-based preprocessing phase. All strategies are evaluated in an extensive experimental study on real-world and real-world based instances. The
ILP-based and the learning heuristics perform very well, usually resulting in near-optimal solutions. An additional advantage of the learning heuristics is its simplicity, robustness, and speed. This leads to better practicability in real-world applications. Furthermore, our experiments indicate that the learning-based approach can be enhanced to also suggest good priority decisions in case of capacity constraints.

Finally, we gain interesting new insights into the structure of good solutions: Firstly, single wait-depart decisions have in real-world railway data a surprisingly local impact on the solution. Secondly, optimal solutions mostly stick to a decision rule which is astonishingly simple applicable once its parameters have been identified.
References

Verzeichnis der erschienenen Preprints 2011:

2011-01  M. Braack, G. Lube, L. Röhe  Divergence preserving interpolation on aniso-
 tropic quadrilateral meshes

2011-02  M.-C. Körner, H. Martini, A.  Minsum hyperspheres in normed spaces
          Schöbel

2011-03  R. Bauer and A. Schöbel  Rules of Thumb – Practical Online-Strategies for
           Delay Management