Fast Heuristics for Delay Management with Passenger Rerouting

Twan Dollevoet   Dennis Huisman

Econometric Institute and ECOPT, Erasmus University Rotterdam
P.O. Box 1738, NL-3000 DR, Rotterdam, The Netherlands
{dollevoet, huisman}@ese.eur.nl

Process quality & Innovation, Netherlands Railways
P.O. Box 2025, NL-3500 HA, Utrecht, The Neherlands

Econometric Institute Report EI2011-35

October 2011

Abstract

Delay management models determine which connections should be maintained in case of a delayed feeder train. Recently, delay management models are developed that take into account that passengers will adjust their routes when they miss a connection. However, for large-scale real-world instances, these extended models become too large to be solved with standard integer programming techniques. We therefore develop several heuristics to tackle these larger instances. The dispatching rules that are used in practice are our first heuristic. Our second heuristic applies the classical delay management model without passenger rerouting. Finally, the third heuristic updates the parameters of the classical model iteratively. We compare the quality of these heuristic solution methods on real-life instances from Netherlands Railways. In this experimental study, we show that our iterative heuristic can solve large real-world instances within a short computation time. Furthermore, the solutions obtained by this iterative heuristic are of good quality.

Keywords: Public Transportation, Railway Operations, Delay Management, Passenger Rerouting

1 Introduction

Most regular train passengers will recognize the frustration of missing a connecting train when their feeder train arrives at the transfer station with a small delay. In low-frequency railway systems, missing a connection can have a severe impact on the travel time of the passengers, even if the delay of the incoming train is only small. In such cases, it might be better to delay the connecting train slightly. By delaying the connecting train, passengers from the delayed train are able to transfer to the connecting train and do not have to wait for the next train. A train waiting for passengers from a delayed feeder train reduces the punctuality, which is the main performance indicator for most railway operators. However, it improves the reliability of the system as a whole and thereby increases passenger satisfaction. Netherlands Railways, the largest passenger operator in the Netherlands, endorses the importance of the reliability of the railway system and has recently introduced the *passenger punctuality* as a performance indicator. The passenger punctuality measures the ratio of passengers who arrive at their destination with a delay smaller than a certain threshold value.

In railway operations management, determining which connections to maintain in case of a delayed feeder train is the subject of *delay management*. Delay management thus decides which trains should wait for a delayed feeder train and which trains should depart on time. The aim is to minimize the total delay for the passengers. Deciding on the wait-depart decisions is a complex problem: If a train waits for a delayed feeder train, passengers in that train will arrive with a delay at the next station, where subsequent transfers take place. This shows that the effects of the wait-depart decisions propagate throughout the entire railway network. Therefore, when solving the delay management problem, the entire railway network should be considered at once. Nevertheless, the current practice at most railway operators is to apply simple rules of thumb to determine which trains should wait. As an example, Netherlands Railways applies a so-called Waiting Time Rule. For each connection, a threshold is determined. If the delay of the incoming train is smaller than the threshold, the connecting train waits. Otherwise, the train departs on time. Kliewer and Suhl (2011) evaluate a wide range of such simple dispatching rules.

In this paper, we consider *off-line* delay management. In the off-line delay management problem, all primary delays in the system are known before the optimization process starts. Off-line delay management is useful when delays can be predicted, for example when construction works restrict the maximal speed of the trains. Furthermore, when a set of primary delays is known in the system, the secondary delays can be determined easily. Off-line delay management can then propose how to react to these secondary delays. Finally, solution methods for off-line delay management can be applied to solve on-line problems in a real-time setting. This application requires solution methods that solve the off-line delay management within a short computation time. The delay management problem that we consider is: Given a set of source delays in a railway system, determine a *disposition* timetable with a new set of maintained connections, such that the total delay of the

passengers is minimized.
A crucial aspect in delay management is to determine the delay for the passengers. To evaluate the delay for passengers who miss a connection, one has to determine how passengers react if a connecting train is missed. Early delay management models assume that such passengers wait for the next train on the same line (Schöbel, 2001). The delay can then be approximated by the cycle time of the timetable. This approximation is correct if the never-meet-property holds (Schöbel, 2007). However, in dense railway networks such as in the Netherlands, this property is mostly not respected. Gatto et al. (2005) show that the general delay management problem is NP-hard, even under the assumption that passengers wait a complete cycle time. To overcome the difficulty of computing the delay in the case of missed connections, Ginkel and Schöbel (2007) consider the delay management problem as a bi-criteria problem, and optimize the delay of the trains and the number of missed connection simultaneously.
The delay management models mentioned so far ignore the limited infrastructure capacity of the railway system: The limited number of tracks and platforms are not taken into account. In order to obtain solutions to the delay management problem that are feasible in practice, one should consider the infrastructure capacity explicitly. A first approach to take the limited capacity of the tracks into account is presented in Schöbel (2009). Computational tests and heuristics are described in Schachtebeck and Schöbel (2010). In Dollevoet et al. (2011b), the limited capacity of the stations is taken into account. Another line of research originates from the conflict detection and resolution literature (D'Ariano et al., 2008), which takes a microscopic view of the railway network to find conflict-free dispostion timetables. In Corman et al. (2010), this microscopic view is applied to minimize the train delay and number of missed connection simultaneously.
Another extension to the classical delay management problem concerns the computation of the delay for passengers who miss a connection. Dollevoet et al. (2011a) compute this delay more realistically: It is assumed that passengers who miss a connection will take the fastest alternative route to their destinations. To do so, they include the routing decisions of the passengers in the integer programming formulation from Schöbel (2007). In this way, a route is determined for all passengers explicitly. In an experimental study, it is shown that determining the routing decisions already during the optimization of the wait-depart decisions reduces the delay for the passengers significantly. However, for large-scale real-world instances, the integer programs become too large to be solved by standard integer programming techniques.
In this paper, we propose several heuristic solution approaches for the delay management problem that incorporate the routes of the passengers. Our aim is to find solution methods that balance the computation time on one hand and the quality of the solution on the other hand. We will compare

the heuristic solutions to the ones obtained by the exact algorithm for the delay management problem with passenger rerouting and to solutions that are obtained by simple dispatching rules. We will show that a solution can be found for large-scale real-world instances in a reasonable amount of time without compromising the solution quality too much.
The remainder of this paper is organised as follows. In Section 2, we describe the delay management model and review the integer programming formulation for delay management with passenger rerouting. Section 3 introduces the heuristic solution methods that we consider. In Section 4 we describe our experimental setup and compare the various solution methods. Finally, in Section 5 we draw some conclusions and discuss possibilities for further research.

2 Delay Management Model

We will now first describe our delay management model formally and then present an integer programming formulation. The integer programming formulation will be used as a benchmark for our heuristic methods and will serve as the basis for some of the heuristics.
Most approaches model the off-line delay management problem with an event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$. In this directed graph, the events are the departures and arrivals of the trains, that need to be scheduled. We denote the set of arrival and departure events as $\mathcal{E}_{\text{dep}}$ and $\mathcal{E}_{\text{arr}}$, respectively. The events are connected by activities, that represent constraints on the times when these events take place. For example, a waiting activity makes sure that the departure of a train from a station takes place after its arrival there. Driving activities ensure that a minimal driving time between the departure of a train from a station and its arrival at the next station is respected. We denote the waiting and driving activities by $\mathcal{A}_{\text{wait}}$ and $\mathcal{A}_{\text{drive}}$, respectively. The transfers from one train to another are represented by activities that can be removed from the network and denoted by $\mathcal{A}_{\text{change}}$. Only transfers that are maintained pose restrictions on the departure times of the connecting trains. For each activity $a \in \mathcal{A} = \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{change}}$, we denote the minimal time required for that activity by L_a.
The passenger data is represented as a set $\mathcal{P}$ of origin-destination pairs (OD-pairs). Each OD-pair $p \in \mathcal{P}$ represents a group of w_p passengers that want to travel from an origin to a destination at a given time. To determine a route for the passengers explicitly, we solve a shortest-path problem in the event-activity network $\mathcal{N}$. In order to do this, we introduce for each OD-pair an artifical source and sink node in the network. These nodes are referred to as origin and destination events, respectively, and denoted by $Org(p)$ and $Dest(p)$. The source and sink nodes are connected to the regular events by origin and destination arcs. The origin arcs connect the source $Org(p)$ for

an OD-pair p to all departure events $e \in \mathcal{E}_{\text{dep}}$ that correspond to a departure from the origin station of p after the time when the passengers in p start their journey. $\mathcal{A}_{\text{origin}}$ denotes the set of all origin arcs. Similarly, all arrival events at the destination station of OD-pair p are connected to the sink node. $\mathcal{A}_{\text{destination}}$ denotes the set of destination arcs.
Note that the arcs $a \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{change}}$ can be used by any OD-pair p. On the contrary, an origin arc $a = (Org(p), e) \in \mathcal{A}_{\text{origin}}$ can only be used by OD-pair $p \in P$. To ease the notation later on, we denote the set of activities that can be used by an OD-pair $p \in \mathcal{P}$ by

$$\mathcal{A}(p) = \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{change}} \cup \delta^{\text{out}}(Org(p)) \cup \delta^{\text{in}}(Dest(p)).$$

We assume that the original timetable and a set of delays is given. For each event $e \in \mathcal{E} = \mathcal{E}_{\text{dep}} \cup \mathcal{E}_{\text{arr}}$, the time when event e is planned is denoted by π_e. The delay at event e is denoted by d_e. Note that $d_e = 0$ for events $e \in \mathcal{E}$ that are not delayed.
For each changing activity $a \in \mathcal{A}_{\text{change}}$, we now determine whether the corresponding transfer is maintained. In order to do so, we introduce for each $a \in \mathcal{A}_{\text{change}}$ a binary decision variable

$$z_a = \begin{cases} 1 & \text{if connection } a \text{ is maintained,} \\ 0 & \text{otherwise.} \end{cases}$$

For each event $e \in \mathcal{E}$, we determine a new time x_e when event e takes place. The values x_e together determine the disposition timetable. Furthermore, for each OD-pair $p \in \mathcal{P}$, we determine which trains the passengers in that OD-pair take. This corresponds to determining a path in the event-activity network for each OD-pair. To model this, we introduce for each OD-pair $p \in P$ and each activity $a \in \mathcal{A}(p)$ a binary decision variable

$$q_{ap} = \begin{cases} 1 & \text{if OD-pair } p \text{ uses activity } a, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, for each OD-pair $p \in \mathcal{P}$, we introduce an auxiliary variable t_p that represents the arrival time for passengers in OD-pair p. The integer programming formulation for delay management with passenger rerouting reads as follows (Dollevoet et al., 2011a).

$$\min \sum_{p \in \mathcal{P}} w_p t_p \tag{1}$$

such that

$$
\begin{aligned}
x_e &\geq \pi_e + d_e, && \forall e \in \mathcal{E}, && (2)\\
x_e &\geq x_{e'} + L_a, && \forall a = (e', e) \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}, && (3)\\
M(1 - z_a) + x_e &\geq x_{e'} + L_a, && \forall a = (e', e) \in \mathcal{A}_{\text{change}}, && (4)\\
q_{ap} &\leq z_a, && \forall p \in \mathcal{P}, a \in \mathcal{A}_{\text{change}}, && (5)\\
1 &= \sum_{a \in \delta^{\text{out}}(Org(p))} q_{ap}, && \forall p \in \mathcal{P}, && (6)\\
\sum_{a \in \delta^{\text{in}}(e) \cap \mathcal{A}(p)} q_{ap} &= \sum_{a \in \delta^{\text{out}}(e) \cap \mathcal{A}(p)} q_{ap}, && \forall p \in \mathcal{P}, e \in \mathcal{E}, && (7)\\
\sum_{a \in \delta^{\text{in}}(Dest(p))} q_{ap} &= 1, && \forall p \in \mathcal{P}, && (8)\\
t_p &\geq x_e - M(1 - q_{ap}), && \forall a = (e, Dest(p)) \in \mathcal{A}_{\text{destination}}, && (9)\\
x_e &\in \mathbf{N}, && \forall e \in \mathcal{E}, && (10)\\
z_a &\in \{0, 1\}, && \forall a \in \mathcal{A}_{\text{change}}, && (11)\\
q_{ap} &\in \{0, 1\}, && \forall p \in \mathcal{P}, a \in \mathcal{A}(p) && (12)\\
t_p &\in \mathbf{N}, && \forall p \in \mathcal{P}. && (13)
\end{aligned}
$$

The objective function (1) minimizes the weighted sum of realized arrival times. As the planned arrival times are fixed, this is equivalent to minizing the weighted sum of delays. Constraints (2) make sure that no event e takes place earlier than planned and that the source delays are taken into account. (3) propagate delays along waiting and driving activities a, while (4) propagate them along *maintained* changing activities a. Constraints (5) state that passengers of OD-pair p can only use a changing activity a if the corresponding transfer is maintained. Equations (6)-(8) formulate a shortest-path problem for each OD-pair p. Finally, Constraints (9) compute the arrival times of the passengers. The constant M in (4) and (9) is a sufficiently large number. Dollevoet et al. (2011a) propose a finite value for M that is large enough.

3 Heuristic solution approaches

In this section we will describe several heuristic methods to solve the delay management problem. Recall from Section 1 that the delay management problem asks for wait-depart decisions, a disposition timetable and new routes for the passengers. At this stage we should note that if the wait-depart decisions are given, finding the optimal timetable and routes is trivial. The disposition timetable can be found by the critical path method (Schöbel, 2007), while the routes for the passengers can be found by solving a shortest-path problem in the event-activity network. This implies that every method

that determines the wait-depart decisions can be used as a heuristic.
An easy policy to implement in practice is not to consider the connections at all. In that case, delays are propagated throught the network only by the driving and waiting activities. As a consequence, all trains depart as early as possible. We call this heuristic a no-wait policy (NW). We will use this heuristic as a benchmark for all the methods that we develop next.

3.1 Simple dispatching rules

In the current operations, railway traffic controllers usually apply simple dispatching rules to determine whether or not to maintain a connection. In Kliewer and Suhl (2011), a large set of such dispatching rules is compared. We have implemented two rules to compare our heuristic solutions to. The first one, the Waiting Time Rule (WTR), corresponds to the current practice at Netherlands Railways.
Under a WTR policy, a maximal waiting time is determined for each connection. This maximal waiting time can be differentiated for different types of connections or even for each connection individually. However, we will assume a constant time $d_{\max}$ for all connections for simplicity. Let now a changing activity $a = (e, e') \in \mathcal{A}_{\text{change}}$ be given and assume that the arrival event e is delayed. It is easy to determine the time that event e' has to be delayed in order to maintain the connection. Denoting this delay to maintain connection a by d_a, it holds that

$$d_a = \pi_e + d_e + L_a - \pi_{e'}.$$

A connection a is maintained if $d_a \le d_{\max}$ and dropped otherwise. We have experimented with different values of $d_{\max}$. Note that $d_{\max} = 0$ corresponds to a no-wait policy.

The drawback of the previous dispatching rule is that it does not take into account the number of transferring passengers. To improve the solutions, the second rule that we consider is the Ratio of Transferring Passengers (RTP). For each connection, first the number of passengers who use a connection is determined. This number is then compared to the number of passengers that have planned to take the connecting train. For a connection $a = (e', e) \in \mathcal{A}_{\text{change}}$, we thus compute the following ratio

$$\rho_a = \frac{\text{Number of passengers that } \textit{planned} \text{ to use connection } a}{\text{Number of passengers that } \textit{planned} \text{ to use driving activity } (e, f)}.$$

Note that for each departure event e, there is exactly one driving activity $(e, f) \in \mathcal{A}_{\text{drive}}$. If the ratio of these numbers is larger than a certain threshold $\rho_{\min}$, *i.e.*, $\rho_a \ge \rho_{\min}$, then connection a is maintained. Again, for values $\rho_{\min}$ larger than 1, we obtain the no-wait policy.

3.2 The classical model as a heuristic

The next heuristic method applies the delay management model from Schöbel (2007). Recall from the introduction that this model assumes that passengers wait for a complete cycle time when they miss their connection. As input, the model needs the number of passengers w_e that plan to end their journey at event $e \in \mathcal{E}_{\text{arr}}$ and the number of passengers $a \in w_a$ that use a connection $a \in \mathcal{A}_{\text{change}}$. Given the set of OD-pairs $\mathcal{P}$, these numbers can be computed as

$$w_e = \sum_{p \in \mathcal{P}(e)} w_p, \qquad w_a = \sum_{p \in \mathcal{P}(a)} w_p,$$

where $\mathcal{P}(e)$ denotes the set of OD-pairs that planned to arrive at their destination with event e and $\mathcal{P}(a)$ denote that set of OD-pairs that planned to use a connection $a \in \mathcal{A}_{\text{change}}$. Denoting T for the cycle time of the timetable, the delay for the passengers is approximated by

$$\sum_{e \in \mathcal{E}_{\text{arr}}} \sum_{p \in \mathcal{P}(e)} w_p x_e + \sum_{a \in \mathcal{A}_{\text{change}}} \sum_{p \in \mathcal{P}(a)} w_p T(1 - z_a). \tag{14}$$

An integer programming formulation for the classical model is now given by the objective function (14) together with the constraints (2)-(4) and (10)-(11). Note that passengers who miss a connection are counted twice in the above objective function: Both the train they planned to arrive with is included in the first term, and the transfer that they missed is included in the second term. The model is correct if the so-called never-meet-property holds (Schöbel, 2007), but for dense railway networks as the one in the Netherlands, this property is often not satisfied.
The classical model assumes that passengers who miss a connection have to wait for one cycle time T, and therefore adds a penalty T to the objective function for every connection that is dropped. Instead of assuming that passengers wait for one cycle time, we can also assume that there is an estimate D of the *additional* delay for passengers who miss a connection, and add this estimate as a penalty to the objective function for every missed connection. The objective function then becomes

$$\sum_{e \in \mathcal{E}_{\text{arr}}} \sum_{p \in \mathcal{P}(e)} w_p x_e + \sum_{a \in \mathcal{A}_{\text{change}}} \sum_{p \in \mathcal{P}(a)} w_p D(1 - z_a).$$

We now view the estimate D as a parameter and find the best value D^* by trial and error. Note that the classical model gives both the wait-depart decisions and the disposition timetable. Given the disposition timetable, it is easy to determine the fastest path for each OD-pair $p \in \mathcal{P}$ in order to evaluate the total delay.

3.3 An iterative heuristic

The previous set of heuristics assumes a fixed estimate D for the additional delay: This delay is equal for all OD-pairs. We will now relax this assumption, and allow for an estimate D_p that differs among the OD-pairs $p \in \mathcal{P}$. Given a set of values D_p, we can solve the classical delay management model with objective function

$$\sum_{e \in \mathcal{E}_{\text{arr}}} \sum_{p \in \mathcal{P}(e)} w_p x_e + \sum_{a \in \mathcal{A}_{\text{change}}} \sum_{p \in \mathcal{P}(a)} w_p D_p (1 - z_a).$$

We will find the best value for D_p with an iterative approach. Given the values D_p, we can solve the classical delay management model. With the wait-depart decisions and disposition timetable that we then obtain, we can determine new routes for the passengers. If we find an OD-pair that misses a connection, we can compute the actual delay for this OD-pair, and use this to update the estimate D_p. This process can be repeated until the best values D_p are found. A more formal outline of the iterative algorithm is given below.

1. Initialize: Set $D_p = 0$ for all $p \in \mathcal{P}$.
2. Repeat until convergence or until a maximum number of iterations has been reached:
   (a) Solve the classical delay management model with the current values D_p.
   (b) Compute the fastest routes for the passengers. This gives the additional delay d_p for an OD-pair p that misses a connection.
   (c) Update the values D_p for all passengers who miss a connection:
   $$D_p = d_p - x_e,$$
   where e is the arrival event that OD-pair p planned to arrive with.

In the initialization step, all values D_p are set to 0. This means that passengers are not delayed if they miss a connection. In particular, the first solution of the classical delay management model will drop all connections and the timetable that we then find will equal that of a no-wait policy. For the passengers in an OD-pair $p \in \mathcal{P}$ who miss a connection in this no-wait policy, the value D_p will be updated in Step 2c. Recall that an OD-pair p is counted twice in the objective function. D_p should therefore be an estimate of the *additional* delay of OD-pair $p \in \mathcal{P}$. This explains the term x_e in the update of the estimates in the final step. In the subsequent iterations, the classical model will find a balance between this additional delay for OD-pair p and the delay that is obtained when the connection is maintained.

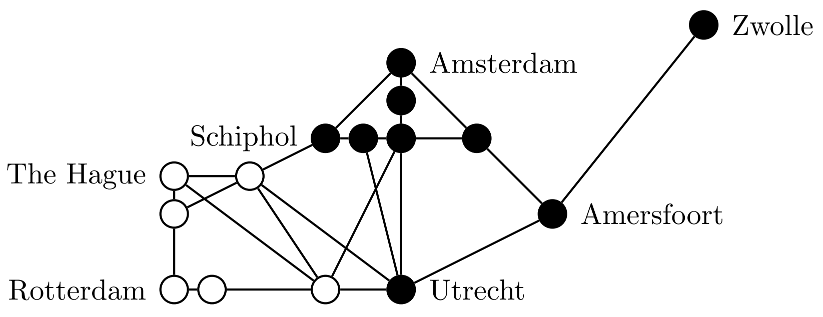

Figure 1: The railway network that is considered in the numerical experiments

4 Numerical experiments

We have performed an experimental study to compare the peformance of the heuristics that are described in the previous section. We will now first describe our experimental setup and then discuss the results.

4.1 Cases

We obtained detailed passenger data from Netherlands Railways, the largest passenger operator in the Netherlands. We have created six real-world instances of varying size to determine how the solution time and solution quality depend on the size of the networks and on the number of passengers. Five of these instances are also used in Dollevoet et al. (2011a).

In Figure 1, the railway network that we consider is depicted. The graph represents part of the Dutch railway network in the mid-Western part of the country. It contains a node for each transfer station in the network. Two nodes are connected by an edge if there is a train service between the stations. On most links, both a long distance and a regional train service is operated, typically with a frequency of two trains per hour. Note that there can be many smaller stations on a link, that are serviced by the regional trains only.

All cases consider a period of four hours in the evening. In the first three cases, we consider all stations indicated by a black dot in this figure. The first case includes all long distance trains. As passenger weights, we consider for each OD-pair the *average* number of passengers. The second and third case consider both the long distance and the regional trains on this network. In the second case, we again take the average number of passengers as the weights. Recall however that the regional trains stop at all stations in the network. As a consequence, the number of OD-pairs is much larger than in the first case. For many of them, the average number of passengers is relatively small. In order to deal with this enormous amount of OD-pairs, in

Case	Stations	Trains	OD-pairs	Passengers	Departures	Transfers
I	10	117	355 (36%)	147 (12%)	219	1074
II	34	284	3940 (65%)	261 (12%)	1022	8068
III	34	284	908 (28%)	289 (12%)	1022	8068
IV	16	168	914 (55%)	345 (21%)	349	1723
V	82	404	22256 (81%)	705 (17%)	2053	13812
VI	82	404	2875 (39%)	775 (17%)	2053	13812

Table 1: Some characteristics of the instances

the third case we consider a possible realization of passenger figures for one day. The realization is constructed in such a way that the expectation of the passenger figures equals the average. We refer to Dollevoet et al. (2011a) for more details on the process of generating the realizations of the passenger figures.

The remaining three cases consider the entire network in Figure 1. The fourth case includes only the long distance trains. The fifth case considers also the regional trains. Both cases use the average passenger figures as weights. Finally, the sixth case considers all trains on the network, but considers a possible realization of the passenger demand.

In Table 1, some characteristics of the cases are given. For each case, we list the number of stations, the number of trains, the number of OD-pairs, the number of passengers, and the number of departures and transfers in the event-activity network. The number of passengers has been scaled for secrecy and does not represent the true numer of passengers. In the columns with the number of passengers and the number of OD-pairs, we have indicated in brackets the percentage of passengers that have to transfer during their trip. It can be seen that the percentage of passengers that transfer is much smaller than the percentage of OD-pairs with a transfer. This implies that the passenger weights w_p are much larger for OD-pairs $p \in \mathcal{P}$ that need no transfer.

To evaluate the heuristic methods, we have simulated 100 delay scenarios for each case. The scenarios are constructed as follows. First, each arrival has a probability of 10% to be delayed. Second, if a train is delayed, its delay is a uniformly distributed number between 1 and 15 minutes.

4.2 Dispatching rules

Our first heuristics apply simple dispatching rules to find the wait-depart decisions. These heuristics maintain a connection if a property of this connection exceeds a certain threshold value. We will now show how to find the best values for these threshold values.

$d_{\max}$	I	II	III	IV	V	VI
0	177053	558667	6068	540393	1917679	20736
1	175178	<u>551059</u>	<u>5992</u>	531780	1877530	20293
2	<u>174863</u>	554903	6033	<u>526938</u>	<u>1869495</u>	<u>20206</u>
3	178453	574323	6246	527224	1903082	20578
4	182580	612444	6662	533188	1986597	21501
5	189927	670575	7298	543174	2119217	22969

Table 2: The results of the Waiting Time Rule for the first three cases. For each case, the best result is underlined.

4.2.1 Waiting Time Rule

The Waiting Time Rule (WTR) maintains a connection if the connecting train has to wait at most $d_{\max}$ minutes. To find the best value for the threshold paramater $d_{\max}$, we have evaluated the heuristics for values ranging from 0 to 5 minutes. In Table 2, the results are presented. The first column in this table gives the value of the parameter $d_{\max}$. Then, the average objective value is given for each of the cases. Recall that for $d_{\max} = 0$, WTR corresponds to a no-wait policy. For the first three cases, with a value $d_{\max} > 2$ for the threshold, the WTR heuristic performs worse than the no-wait policy. It turns out that allowing all connecting trains to wait for more than 2 minutes for a delayed feeder train gives a bad policy. For Cases IV and Cases V and VI, WTR performs worse than the no-wait policy if $d_{\max} \geq 5$ and $d_{\max} \geq 4$, respectively. In all cases, the WTR peforms better than a no-wait policy for $d_{\max} \in \{1, 2\}$: The delay is reduced on average by 1.9%. For the second and third case, the best results are found for $d_{\max} = 1$. For all other cases, $d_{\max} = 2$ gives the best performance.
For easier comparison of the results among the cases, we have normalized the objective value and plotted these relative objective values in Figure 2. The objective value for the no-wait policy is set to 100 percent. For values of $d_{\max} > 0$, we have computed the delay relative to the no-wait policy. One immediately notices that the behavior of the policy is very similar for Cases II and III, and for Cases V and VI. Recall that these cases consider the same railway system, and differ only in the OD-pairs that are considered. It follows from this graph that a small subset of OD-pairs can be used to evaluate the performance of the heuristic for the case that includes all OD-pairs.

4.2.2 Ratio of Transferring Passengers

The Ratio of Transferring Passengers (RTP) heuristic computes the ratio of the number of passengers who plan to use a connection and the number of passengers who plan to use the connecting train. If this ratio exceeds a given

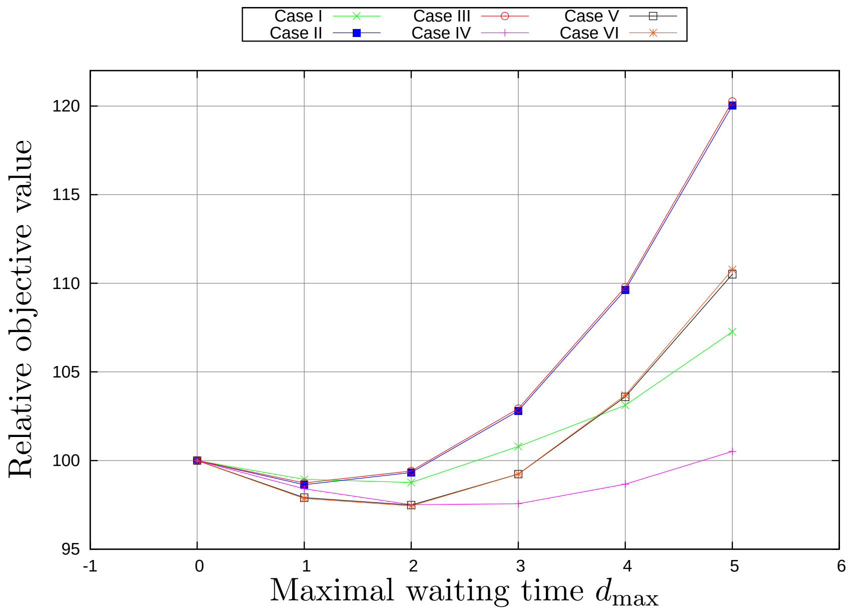

Figure 2: The results of the Waiting Time Rule for Cases IV-VI

threshold $\rho_{\min}$, the connection is maintained; otherwise it is dropped. We experimented with values of $\rho_{\min}$ between 0 and 100 percent. Note that we obtain a no-wait policy if $\rho_{\min} > 100\%$. In order to compare the results to the no-wait policy, we therefore also applied the heuristic with $\rho_{\min} = 110\%$.

In Figure 3, the results of the RTP heuristic are presented for all cases. From the graph we see that with the best value for the parameter $\rho_{\min}$, the RTP dispatching rule performs about 4% better than the no-wait policy. Contrary to the WTR rule, the best value for the parameter $\rho_{\min}$ differs among the cases. For Cases I and IV, that consider only the long distance trains, the best value is found at 30% and 20%, respectively. For the other cases, that include also the regional trains, the best heuristics are found with $\rho_{\min}$ equal to 40% or 50%. We conclude that if both long distance and regional trains are considered, the optimal ratio $\rho_{\min}$ should be higher. In order to explain this, recall that the regional trains stop at all stations along the railway line. Consider a transfer to a regional train that travels in the direction of a larger stations and stops at some smaller stations along the way. Many passengers will enter the regional train in one of the smaller stations and travel towards the larger station. When we determine whether to maintain the connection at the transfer station or not, these passengers are not considered, as they will enter the train at a later time. However, if the connection is maintained, they will be delayed. The RTP heuristic then underestimates the delay that arises if the connection is maintained. Choosing a higher value for $\rho_{\min}$ makes up for this underestimation.
With the optimal value for the parameter, the RTP rule reduces the delay

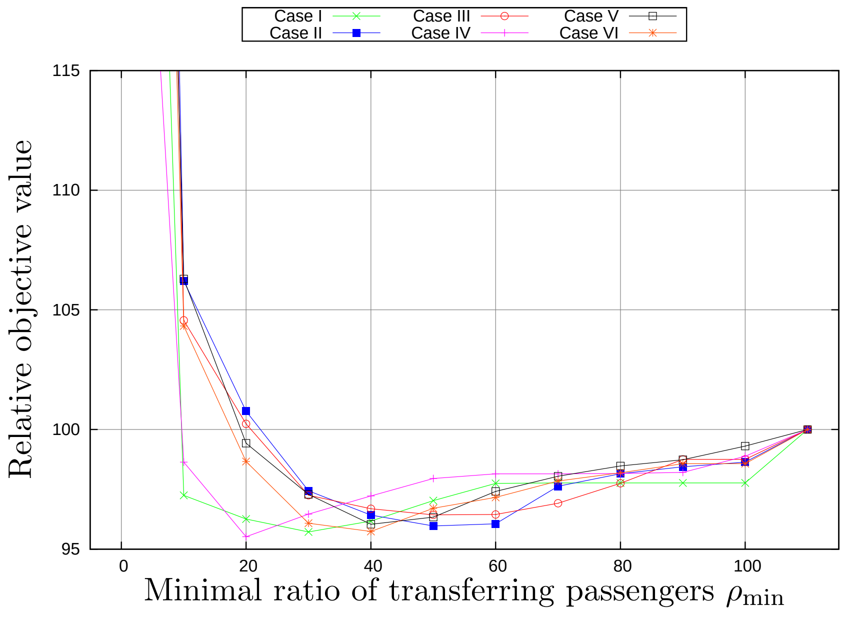

Figure 3: Results for the Ratio of Transferring Passengers rule

roughly by 4%. Comparing the RTP rule to the WTR rule, we see that the performance of the RTP rule is much better. It thus pays off to compare the number of passengers that want to use a connection to the number of passengers that are delayed if the connecting train waits.

4.3 Classical Delay Management

Our second set of heuristics applies the classical delay management model, but views the penalty D in the objective function as a parameter. In the original model, this parameter was set to T, the cycle time of the timetable. If we set $D = 0$, the heuristic equals the no-wait policy. That allows us again to normalize the objective value, in order to be able to compare the results among the cases. We have experimented with values for D ranging from 0 to 60 minutes. Note that $T = 60$ minutes in our timetable.

In Figure 4, the results for all cases are presented. To find the best value for the paramater D, we have evaluated the heuristic for more values around $D = 20$. For all cases, we found that $18 \leq D^* \leq 21$. Furthermore, the differences in objective value are very small among these values. We conclude that the additional delay for passengers that miss a connection is about 20 minutes. The performance of the heuristic with the best value for the parameter D is much better for the last three cases. For the smaller network, the quality of the solutions is 6% better than a no-wait policy. For the larger network, the objective values are reduced by 9%.

Considering the shape of the graph for each case individually, one sees that

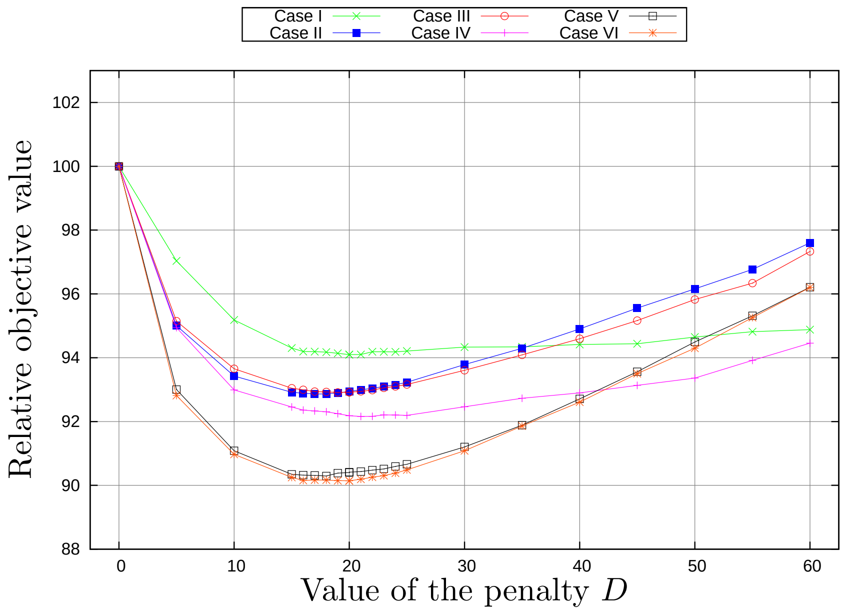

Figure 4: The results for the heuristic that applies the classical model

the graph for Cases I and IV is very flat at the end. If the value of D is increased, this does not give objective values that are much worse. On the contrary, for the other cases the objective value increases if the value of the parameter is inreased. Furthermore, we again see that the results are very similar for Cases II and III, and for Cases V and VI. This suggests that one can use a small set of OD-pairs to find the best value of the parameter D.

4.4 Iterative heuristic

Our final heuristics applies the classical delay management model in an iterative fashion. Contrary to the previous heuristic, it uses a parameter D_p that differs among the OD-pairs $p \in \mathcal{P}$. In each iteration, the heuristic first solves the classical model and then reroutes the passengers. It stops when the method converges or when the maximal number of iterations has been reached. In Table 3, we present the number of iterations that the heuristic needs. The first row gives the number of instances that did not converge. The second row gives the average number of iterations among the instances that did converge. We see in the table that the heuristic converges for almost all instances. Only in Cases III and V, there is one instance that does not converge. For these instances, the heuristic cycles between two solutions, that have almost identical objective values. In both instances, there is one OD-pair that is rerouted in one solution, and takes the planned route in the other. On the other instances, the iterative heuristic converges on average

	I	II	III	IV	V	VI
Instances that did not convergence	0	0	1	0	1	0
Average number of iterations	2.23	3.48	3.09	2.78	3.90	4.24

Table 3: The number of instances for which the iterative heuristic did not converge and the average number of iterations

	I	II	III	IV	V	VI
No-wait	177053	558667	6068	540393	1917679	20736
WTR	174863	551059	5992	526938	1869495	20206
RTP	169477	536135	5852	516173	1841761	19853
Classical	166601	518809	5638	498031	1731831	18692
Iterative	165610	515573	5600	496529	1720748	18534
Exact	165110	512878	5567	495111	-	18343
No-wait	100.0	100.0	100.0	100.0	100.0	100.0
WTR	98.8	98.6	98.7	97.5	97.5	97.4
RTP	95.7	96.0	96.4	95.5	96.0	95.7
Classical	94.1	92.9	92.9	92.2	90.3	90.1
Iterative	93.5	92.3	92.3	91.9	89.7	89.4
Exact	93.3	91.8	91.7	91.6	-	88.5

Table 4: The absolute and relative objective value for each heuristic

in less than five iterations.

4.5 Comparison of the heuristics

In the previous sections we have shown how to find the best parameters for the simple dispatching rules and for the heuristic based on the classical model. We will now compare both the quality and the running time of the heuristics to each other and to the exact approach. In Table 4, the best objective value is presented for each heuristic and for each case. The upper rows contain the absolute objective values, the lower ones contain relative objective values. Again, we used the no-wait policy to normalize the objective values.

We see in the table the simple dispatching rules give the worst results. Although they are easy to implement, the quality of the solutions they produce is bad. Using the classical model as a heuristic, by viewing the penalty D as a parameter, reduces the delay on average by 8%. The iterative heuristic improves slightly over the classical model, reducing the delay by an additional 0.5%. For the first five cases, the problem could also be solved by the exact algorithm. Comparing the objective values obtained with the iterative

	I	II	III	IV	V	VI
Classical	0.04	0.5	0.25	0.08	3	0.5
Iterative	0.05	1	0.3	0.11	6.1	1
Exact	0.48	357	10	1.4	-	62

Table 5: The average running times for the classical, iterative and exact algorithm, in seconds

and exact algorithm for those cases, we see that the iterative heuristic finds solution that are close to optimal. The relative deviation is on average only 0.4%.

Our aim with the off-line delay management heuristics is to apply them in an algorithm for the on-line delay management problem. As the on-line delay management problem should be solved in a short computation time, we are also interested in the running times for the different heuristics. For the heuristics that apply simple dispatching rules, the running times are neglectable. For each case, the running time was so short that it could not reliably be measured. We thus compare only the running times for the classical and iterative heuristic and for the exact algorithm. In Table 5, these average running times are reported.

The running times for the classical and iterative model are very short. Even the largest case can be solved within seconds. For the larger cases, the classical heuristic is about two times faster than the iterative heuristic. The exact algorithm needs much more computation time. It takes on average almost six minutes to solve Case III. For the real-time setting of delay management, such computation times are too long. The running times of the classical and iterative heuristic allow a real-time application.

5 Conclusions

To evaluate the quality of any delay management policy, the routes that passengers choose have to be taken into account. This rerouting aspect of delay management should be considered when solution methods for the delay management problem are developed. In this paper, we have constructed several heuristic methods to solve the off-line delay management problem. We have compared these heuristic among each other and to the exact algorithm in an experimental study. In this study, it is shown that an iterative heuristic, that solves the classical delay management model iteratively, leads to good solutions. On average, the performance of the iterative algorithm is only 0.4% worse than the exact algorithm. This decrease in quality is compensated by the running time of the method; the iterative approach could solve all instances within seconds. We have also implemented simple dispatching

rules, that are currently used in practice. For example, the Waiting Time Rule is applied at Netherlands Railways. In our numerical experiments, it is shown that these dispatching rules do not perform well.
Our delay management model ignores the limited capacity of the railway infrastructure. To obtain solutions that can be implemented in practice, these capacity considerations should be incorporated in the delay management models with passenger rerouting. An exact algorithm for delay management models that includes both passenger rerouting and the capacity constraints might be too ambitious. However, any algorithm for the capacitated delay management problem without rerouting can be applied in an iterative fashion to obtain a heuristic for the capacitated delay management problem with passenger rerouting. Our results on the uncapacitated delay management problem suggest that the total delay for the passengers can then be reduced significanly.

References

F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo. Bi-objective conflict detection and resolution in railway traffic management. *Transportation Research Part C*, 2010. in press.

A. D'Ariano, F. Corman, D. Pacciarelli, and M. Pranzo. Reordering and Local Rerouting Strategies to Manage Train Traffic in Real Time. *Transportation Science*, 42(4):405–419, 2008.

T. Dollevoet, D. Huisman, M. Schmidt, and A. Schöbel. Delay management with re-routing of passengers. *Transportation Science*, 2011a. in press.

T. Dollevoet, M. Schmidt, and A. Schöbel. Delay management including capacities of stations. In A. Caprara and S. Kontogiannis, editors, *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 88–99. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011b.

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

A. Ginkel and A. Schöbel. To wait or not to wait? The bicriteria delay management problem in public transportation. *Transportation Science*, 41(4):527–538, 2007.

N. Kliewer and L. Suhl. A note on the online nature of the railway delay management problem. *Networks*, 57(1):28–37, 2011.

M. Schachtebeck and A. Schöbel. To wait or not to wait and who goes first? Delay management with priority decisions. *Transportation Science*, 44(3): 307–321, 2010.

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

A. Schöbel. Integer programming approaches for solving the delay management problem. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 145–170. Springer, 2007.

A. Schöbel. Capacity constraints in delay management. *Public Transport*, 1 (2):135–154, 2009.