The Computational Complexity of Delay Management

Technical Report 456, Department of Computer Science, ETH Zurich

Michael Gatto* Riko Jacob* Leon Peeters* Anita Schöbel†

Abstract

Delay management for public transport consists of deciding whether vehicles should wait for delayed transferring passengers, with the objective of minimizing the overall passenger discomfort. We model the underlying transportation network as a directed acyclic graph, where edges represent trains, and weighted paths represent passenger flows. Given initial delays of some of the passenger paths, our goal is to decide which edges wait for delayed passenger paths, such that the sum of all passenger delays is minimized.

This paper classifies the computational complexity of delay management problems with respect to various structural parameters, such as the maximum number of passenger transfers, the graph topology, and the capability of edges to reduce delays. Our focus is to distinguish between polynomially solvable and NP-complete problem variants. To that end, we show that even fairly restricted versions of the delay management problem are hard to solve.

1 Introduction

Even a carefully planned railway system will once in a while have to deal with delayed trains due to unforeseeable events. In such a case, the railway operator can react by maintaining some connections and modifying the schedule accordingly.

This paper considers the impact of such modifications on the overall passenger delay. The problem of managing delayed trains is still not well understood, even though the first research on railway delays started as early as two decades ago (see, for example, [HK81]). In particular, no efficient exact algorithms are known so far for any general problem setting. We present an explanation for this situation by showing that several restricted versions of the delay management problem are NP-complete. We identify various combinatorial aspects that cause the problem to be difficult to solve, and complementarily describe some polynomial time algorithms. Thus, we establish a fairly precise complexity boundary that depends on structural parameters of the problem instance.

The delay management problem considers a trade-off that is best explained by an example. Consider a passenger in an on-time train, which decides to wait for a delayed feeder train. Although the passenger was traveling on-time, she now faces a delay because of this decision. Moreover, she herself may later miss a connecting train in a subsequent station. Alternatively, had the train not waited, then the connecting passengers in the feeder train would have missed their connection. In particular, they would have had to wait for the next train, thus facing a large delay each. Delay management consists of deciding which connecting trains should wait for which delayed feeder trains, with the objective of minimizing the sum of the delays faced by the passengers.

We model the railway network as a graph, and passenger flows as fixed paths in this graph. In this network, unforeseen events may occur that result in the late arrival of connecting passengers at transfer stations. Given these so-called source delays, our goal is to decide which connecting trains wait for delayed transfer passengers, such that the sum of all passenger delays is minimized. In our opinion, this model captures the key aspects of delay management, such as the propagation of delays through the network. And because of its abstractness, the model is also applicable to other modes of scheduled public transport. Still, some important real-life aspects are not included, such as the availability of track capacity to accommodate the adjusted schedule.

*Institute of Theoretical Computer Science, ETH Zurich, {gattom,rjacob,peetersl}@inf.ethz.ch

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

Network topology	≤ 2 transfers	No slack times ≤ 3 transfers	< ∞ transfers	With slack times ≤ 2 transfers	< ∞ transfers
General	min cut	*NP-complete*	*NP-complete*	*NP-complete*	*NP-complete*
Series-parallel	min cut	*NP-complete*	*NP-complete*	*NP-complete*	*NP-complete*
Tree	min cut	dynamic program	dynamic program	?	*NP-complete*
Line	min cut	dynamic program	dynamic program	?	*NP-complete*
Tree, single destination				*"pedal-to-the-metal"*	

Table 1: Classification of the binary delay management problem, with implied entries greyed out. Contributions of the paper are in italic.

1.1 Contribution of the Paper

This paper identifies a boundary between NP-completeness and polynomial solvability for various natural problem parameters of the delay management problem. In particular, we focus here on the case where all non-zero source delays are of equal size, which we refer to as binary source delays.

We first show that the binary delay management problem is strongly NP-complete if trains cannot catch up on their delay, already on a railway network with series-parallel topology. As some of the passengers in the complexity reduction transfer three times, the result complements our earlier finding that the problem is polynomially solvable when passengers transfer at most twice [GGJ+04]. We also extend the latter result to the case of unbounded number of transfers, in which initially on-time passengers are not allowed to miss a connecting train (though they are allowed to be delayed).

Next, we study the binary delay management problem with slack times, meaning that trains can catch up on their delay. We show that this variant is already NP-complete on a railway network with a line structure. Again, this contrasts an earlier result on the polynomially solvability of such a line network without slack times [GGJ+04]. Further, a slightly different NP-completeness reduction yields passengers that transfer twice, on a more general network that is series-parallel. As an ingredient for one of our proofs, we establish that the maximum unweighted directed cut problem on directed acyclic graphs is NP-complete.

Without slack times, all source delays must contribute to the objective. For this setting, we also investigate the objective function without this offset, and show that it is NP-hard to approximate to a certain constant factor.

Finally, we describe a polynomial time "pedal-to-the-metal" algorithm for the delay management problem with slack times, under the restriction that all passengers travel to the same destination station on a network with a tree-like structure.

Given our interest in complexity aspects, we focus on fairly simple versions of the delay management problem, which are perhaps not too realistic. Still, our findings give insight into the structure of more complex and more realistic models. Thus, the main contribution of the paper is to identify combinatorial aspects that are crucial for the problem's complexity. Table 1 summarizes the results. Naturally, the unrestricted delay management problem is NP-complete as well.

1.2 Related Research

The above described delay management problem was introduced by [Sch01], who proposed a Mixed Integer Programming formulation for a model that is similar to ours. Schöbel [Sch03] also showed that, when no two delayed vehicles meet in an optimal solution to this model, its constraint matrix is totally unimodular. In that case, an optimal solution can be obtained in polynomial time by Linear Programming. Further, [GGJ+04] described a minimum cut reduction for passengers that transfer at most twice, and a polynomial-time dynamic program for railway networks with a tree topology.

In spite of these algorithmic results, no strong NP-completeness results were known so far for delay management. [Sch03] showed that the the bi-criteria problem of concurrently minimizing the weighted passenger delay and the number of missed connections is weakly NP-complete. For the same bi-criteria problem, [Meg04] provides a slightly different complexity proof and some further theoretical observations.

[GJPW04] provides a first competitive analysis for the on-line version of delay management, on the simplified setting of a single railway line with intermediate stops. For this simple model, a family of 2-competitive algorithms exists, and no on-line algorithm can be more than golden-ratio competitive.

The series of papers [SM01, SBK01, SMBG01, BS04] evaluates deterministic policies for deciding whether trains should wait for delayed transfer passengers. The evaluation of these policies is carried out through an agent-based simulation tool. Since our focus is on optimization and computational complexity, we do not further discuss this and other research on simulation for railway delays, such as [ADGT99, OW99, HHW99].

2 Problem Statement

This section describes the delay management model analyzed in the paper. First, we describe the general model, which is similar to the model in [Sch01]. Next, we specify the considered restrictions of the model.

General Model Definition

The event-activity model in [Sch03] forms the base of our general delay management model. Let $G = (V, E)$ be a directed acyclic graph. Each vertex $v \in V$ represents a station, and each edge $e = (u, v) \in E$ represents a single direct train only operating the connection between u and v. Trains do not have intermediate stops in this model. At each station v, the outgoing edges (v, w) represent the connecting trains for the passengers traveling on the incoming trains (u, v). A directed path in G then corresponds to a journey a passenger can undertake by transferring between trains. We assume transfers to happen instantaneously. Thus, passengers arriving at a station with a delay can only board the connecting train if it waits for the entire delay of the feeder train. Alternatively, one could model the transfers by additional edges in the graph. We omit this construction for simplicity, but point out that these additional edges do not influence our results.

A train $e = (u, v) \in E$ can reduce a possible delay by $\mathcal{S}(e) \geq 0$ time units on its trip from u to v, for example by driving faster than scheduled. We refer to $\mathcal{S}(e)$ as the *slack time* of the train. Trains must not arrive earlier than scheduled, so slack times can only be used if a train departs with a delay.

Passenger flows in the railway system are modeled by a set of directed paths $\mathcal{P}$ in the graph. Such a path P induces transfers at every internal vertex of P. A path $P \in \mathcal{P}$ has an associated weight $w(P)$ representing the number of passengers, or the importance of the path in a more abstract sense. As a direct consequence of an unforeseen event, some passengers may arrive at a transfer station with a delay. In our model, such passengers are represented by a passenger path $P \in \mathcal{P}$ with a source delay $\mathcal{D}(P) > 0$, starting at that transfer station and ending at the passengers' destination. Thus, our model defines source delays on paths rather than on trains. We refer to paths with $\mathcal{D}(P) = 0$ as *source punctual paths*, and to paths with $\mathcal{D}(P) > 0$ as *source delayed paths*.

A passenger path $P \in \mathcal{P}$ misses a connection if it arrives at a transfer station with a delay, and its connecting train does not wait long enough. We assume that trains are operated according to a periodic timetable with period T, and that delays do not propagate to the next period. Examples exist where the latter happens, but we do not consider such cases for the sake of simplicity. Hence, a passenger path $P \in \mathcal{P}$ has an arrival delay $\delta_P = T$ if it misses a connection. If all connections on path P are maintained, its arrival delay δ_P equals the arrival delay of its last train. We refer to paths with $\delta_P = 0$, arriving as scheduled at their destination, as *punctual paths*, to those arriving delayed as *delayed paths*, and to those missing a connection as *dropped paths*. Further, we refer to paths not missing a connection as *maintained paths*. The possibility to drop paths is a key aspect of this setting. Indeed, dropping a path P effectively removes P from the network, such that no train is influenced by P any more.

An instance is completely defined by the tuple $(G, \mathcal{S}, \mathcal{P}, \mathcal{D}, w, T)$. For such an instance, a delay policy π specifies which trains wait, for how long, and how much slack time they use. We wish to find a delay policy π^* that minimizes the *total passenger delay* defined as the weighted sum of arrival delays $\sum_{P \in \mathcal{P}} w_P \delta_P$.

Problem Restricting Parameters

Our complexity results consider several restricted versions of the general model. These restrictions include the basic cases of limiting the maximum number of passenger transfers (maximum passenger path length), restricting the source delays to one single non-zero value (binary source delays), and not allowing for trains to catch up on delays (availability of slack times).

As for the network topology, we consider lines (paths), trees, and series-parallel graphs. Series-parallel graphs have treewidth two, which intuitively means they are almost trees. Many NP-complete problems, such as Independent Set and Vertex Cover, become polynomially solvable on bounded-treewidth graphs. Hence, an NP-hardness result for series-parallel graphs in some sense complements a polynomial time algorithm for trees.

For a discussion of series-parallel graphs and treewidth, see for example [Bod93]. Briefly, the (directed) series-parallel graphs are defined recursively as follows. Every series-parallel graph has a designated source node s and a sink t. The graph consisting of one edge from s to t is a series-parallel graph. Given two series-parallel graphs G and H, their parallel composition and their serial composition are both series-parallel graphs. The parallel composition is obtained by identifying the two source nodes and the two sink nodes. The serial composition is defined by identifying the sink of G with the source of H, and by defining the new source as the source of G, and the new sink as the sink of H.

Remarks on the Model

Note that in one station there may be paths that connect on time as well as paths that connect delayed. Additionally, there may be several paths between one origin destination pair, which can all be used simultaneously by different passengers. This leads to a problem variant with so-called dynamic path choices, in which the optimization includes choosing a path for each passenger.

Dynamic path choices, as used in [GGJ$^+$04], become void if each passenger has a unique path to travel from his origin to his destination. By exploiting this observation, we can strengthen our hardness results to also hold for dynamic path choices, also improving on the result in [GGJ$^+$04] with respect to network topology. Thus, dynamic path choices do not make the problem easier to solve. Actually, considering the inapproximability result of [GGJ$^+$04] for unrestricted topology, it could be that dynamic path choices make the problem harder to solve.

The same holds for other possible restrictions on the passenger paths structure, such as allowing only shortest paths to travel from an origin to a destination.

Finally, consider the natural setting in which a single initially delayed train is the cause of the source delayed paths. Our hardness results can be adapted to this situation by the following modification. Each source delayed path starts with the single delayed train, followed by an artificial train to the path's true origin station. Such an artificial train is defined for each source delayed path, and used uniquely by that path. Thus, a source delayed path travels with the delayed train, and uses the artificial train to reach its true origin station. This construction results in only slightly weaker parameters with respect to the number of passenger transfers and the network topology.

3 Delay Management without Slack Times

Our first analysis considers the restricted setting of binary source delays and no slack times, that is, $\mathcal{D}(P) \in \{0, \delta\}$ for all $P \in \mathcal{P}$, and $\mathcal{S}(e) = 0$ for all $e \in E$. In this setting, an optimal delay policy π^* describes which trains depart on-time, and which ones depart with a delay of size δ. We refer to this restricted model as the binary delay management problem, and write an instance as $(G, \mathcal{P}, \mathcal{D}, w, T)$.

3.1 Proof of Hardness with Three Transfers

In this section, we prove that the binary delay management problem is NP-complete already for unweighted passenger paths on a series-parallel train-network. To that end, we first prove a weaker theorem.

Definition: Decision binary delay management problem.
Instance: A binary delay management instance $(G, \mathcal{P}, \mathcal{D}, w, T)$, $d \in \mathbb{N}$.
Question: Is there a delay policy such that the total passenger delay is less than or equal to d?

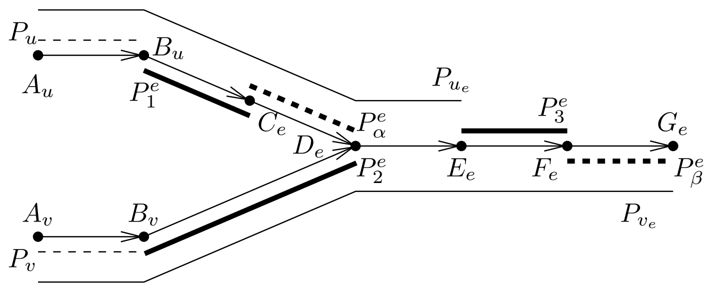

Figure 1: The construction for an extended edge (u, u_e, v_e, v) in G_2. Directed edges represent the trains in the network. Undirected lines represent the paths. Thick paths are the paths of weight M. Dashed paths represent paths with source delay δ.

Theorem 1. *The decision binary delay management problem with passenger paths changing at most four times is NP-complete.*

Proof. It is easy to see that the problem is in NP, as the weighted delay of the paths induced by a delay policy π can be computed in polynomial time, and the size of π is polynomial as well. We show that the problem is NP-hard by reduction from Maximum Independent Set [GJ79, Problem GT20]. Let the undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$ be a Maximum Independent Set instance asking for an independent set of size K. Consider its 2-subdivision [Pol74] $G_2 = (V_2, E_2)$, i.e., the graph obtained by inserting the vertices u_e, v_e for each undirected edge $e = (u, v)$ and splitting the edge into three undirected edges $(u, u_e), (u_e, v_e), (v_e, v)$. We refer to this construction for an edge $e = (u, v)$ of the original graph as the extended edge in the 2-subdivision, symbolized by (u, u_e, v_e, v). The graph G has a maximum independent set of size K if and only if its 2-subdivision G_2 has a maximum independent set of size $K + m$.

In the following, we construct gadgets for every extended edge of the 2-subdivision graph. In the resulting delay management instance, certain paths that are maintained in an optimal delay policy π^* correspond to the vertices in the maximum independent set of the 2-subdivision.

For each vertex q in G_2 we construct a path P_q in the delay management instance, such that two vertices q, r can be in the same independent set if and only if the corresponding paths P_q and P_r can both be maintained in the same optimal delay policy. A maximum independent set in G_2 hence corresponds to an optimal set of maintained paths.

For this construction, consider an extended edge (u, u_e, v_e, v). For the vertices u, v we have paths $P_u, P_v \in \mathcal{P}$, both with unit weight and unit source delay. These paths exist once for every $u \in V$. Further, we introduce paths P_{u_e}, P_{v_e} for u_e and v_e, both with unit weight and no source delay. The exact configuration of all these paths is shown in Figure 1.

For each extended edge (u, u_e, v_e, v) of the 2-subdivision we introduce five paths in the delay management instance, P_1^e, P_2^e, P_3^e, P_α^e, P_β^e, each with weight $w(P_i^e) = M, i \in \{1, 2, 3, \alpha, \beta\}$, where M is a sufficiently large value. The paths P_α^e and P_β^e have source delay δ, the other paths have no source delay. Because of the large weight M, the source delayed paths P_α^e, P_β^e will never be dropped in an optimal delay policy π^*. For the same reason, the paths $P_i^e, i \in \{1, 2, 3\}$ will always be kept punctual. We refer to these paths as M-paths, and Figure 1 shows their exact configuration.

Let π^* be an optimal delay management policy for the constructed instance, meaning that no M-paths are dropped. In π^*, the paths corresponding to vertices of G_2 interact by sharing edges. Because of the M-paths, π^* cannot maintain two such interacting paths, since one requires the shared edge to be delayed, whereas the other requires it to be on-time. Indeed, P_u and P_{u_e} share the edge (A_u, B_u), and P_{u_e} and P_{v_e} share (D_e, E_e). Note that this construction enforces that each maintained unit weight path will be delayed.

Hence, the unit weight paths that are maintained in π^* correspond to an independent set I in G_2. Since every maintained path reduces the cost of the delay policy, I is a maximum independent set.

More precisely, set $\delta = 1$, $T = 2$, $M = m + 2$, and $d = 2mM\delta + (2m + n)T - (m + K)(T - \delta)$. Now G_2 has an independent set of size $m + K$ if and only if the binary delay management instance has a

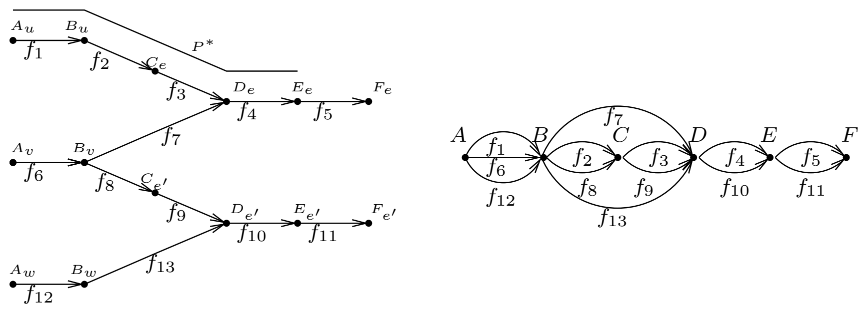

Figure 2: On the left hand side, a network stemming from Independent Set on nodes u, v, w and edges $e = (u, v), e' = (v, w)$. Except for P^*, the paths of the construction are omitted. Capital letters are the node labels. On the right hand side, the corresponding series-parallel network after the contraction has taken place. Note that the path P^* is routed along the edges f_1, f_2, f_3, f_4, as in the uncontracted version.

delay policy π with cost at most d, i.e., which maintains $m + K$ unit weight paths.

Finally observe that the longest constructed passenger path requires 4 changes. $\square$

Theorem 1 can be strengthened by constructing an even simpler instance of the delay management problem.

Theorem 2. *The decision binary delay management problem on a series-parallel graph with passenger paths changing at most three times and unweighted passenger paths is NP-complete.*

Proof. We modify the construction of Theorem 1 as follows. To reduce the maximal number of changes to 3, the last gadget edge (F_e, G_e) is omitted for each $e \in E$. This still enforces that one of the paths P_{v_e} and P_{u_e} must be dropped. With this construction, dropping P_{u_e} and maintaining P_{v_e} causes a delay of T, which is less than the cost $\delta + T$ of maintaining P_{u_e} and dropping P_{v_e}.

Set $\delta = 1, T = m + 2, M = m + 3, w(P_i) = 1, i \in \{u, v, u_e, v_e\}$ and $d = 2n - K + mn - mK + 2m^2 + 6m$. We claim that G_2 has an independent set of size $m + K$ if and only if the modified binary delay management instance has a delay policy with cost at most d, which maintains K unit weight paths P_u and m unit weight paths P_{u_e}. Such a policy contributes to the objective with weight $K\delta$ for the maintained paths P_u, with $(n - K)T$ for the dropped paths of the same type, with $m\delta M$ for the source delayed M-paths, and with weight mT for the dropped paths P_{u_e} and P_{v_e}. Although dropping the paths P_{u_e} and P_{v_e} contributes to the objective with weight T per path, maintaining P_{u_e} has a different impact on the objective than maintaining P_{v_e}. Indeed, maintaining P_{v_e} causes no addition to the objective, whereas maintaining P_{u_e} increases the objective by δ. Over all edges e, this contribution can nevertheless be bounded by $m\delta$. Adding all the above terms yields the value of d.

Delaying or dropping M-paths is more expensive than dropping unit weight paths. Further, dropping more than $n - K + m$ unit weight paths causes the delay to exceed d, as can be verified. Thus, the construction enforces at least $K + m$ paths to be maintained, that is, at most $n - K + m$ paths to be dropped.

To make the underlying network series-parallel, we identify all nodes of one position into one single node. That is, all A_u nodes are contracted into one node A, all B_u nodes into one nodes B, all C_e into C, and so on. Then, the graph consists of the 6 nodes A, B, C, D, E, F, with bundles of parallel edges between (A, B), (B, C), (B, D), (C, D), (D, E), and (E, F), see Figure 2.

In the resulting series-parallel network, each path still uses the same edges as prior to contracting the nodes. For example, the path P^* in Figure 2 uses the edges f_1, f_2, f_3, f_4 in the series-parallel network. The interaction of the gadgets in the series-parallel network is achieved by paths that share an edge.

Observe that the weight $M = m + 2$ is polynomial in the size of the instance. Thus, as a last step, we can transform the reduction to an instance with unweighted paths by introducing M parallel passenger paths of unit weight for each M-path. $\square$

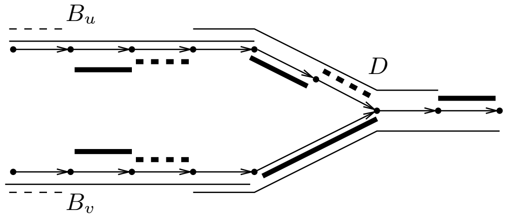

Figure 3: The non-contracted graph with dynamic path choices before the contraction and the embedding into a series-parallel network.

3.2 Dynamic Path Choices

Unfortunately, dynamic path choices render the construction in the proof of Theorem 2 useless. To see this, note that all paths with the same "function" in the construction of Theorem 1 start and end at the same nodes in the above series-parallel network. For example, all paths $P_v, v \in V$, start at A and end at B, and all paths $P_v^e, e = \{u, v\} \in E, v \in V$, start at A and end at F.

Consider the following solution. For each bundle of parallel edges, set one edge to be punctual, one edge to be delayed, and all other edges arbitrarily. Now, each source punctual path can dynamically choose the punctual edges, and each source delayed path can dynamically choose the delayed edges. Clearly, this solution does not drop any paths, nor does it delay any source punctual paths.

As a remedy, the construction can be modified such that the route of each passenger path is unique in the series-parallel network. The construction uses a 4-subdivision instead of a 2-subdivision. To this aim, each of the $2m$ parallel subgraphs between B and D is replaced by a serial interaction gadget, with one on-time M-path and one delayed M-path. Further, the paths between B and D in the original construction are split, see Figure 3. In this modified construction, only the vertices B_u and D_e are contracted. This construction can be embedded in a series-parallel graph. The resulting graph before the contraction is depicted in Figure 3. As a result, after contracting the vertices B_u and D_e, each path has a unique feasible route.

Corollary 3. *The decision binary delay management problem on a series-parallel graph with passenger paths changing at most three times and unweighted passenger paths is NP-complete, even with dynamic path choices.*

3.3 Approximating the Additional Delay

As stated in Section 2, our objective is to minimize the total passenger delay. Alternatively, it also makes sense to minimize only the weighted delay that paths face in addition to their source delay. Indeed, as there are no slack times, a source delayed path can never do better than arrive at its destination with a delay of δ. This portion of the delay cannot be optimized, so it is reasonable to omit it from the objective function. We refer to this alternative objective function as the additional weighted delay, which should be minimized.

As Independent Set and Vertex Cover are complementary problems, the results from [Hås01] provide an inapproximability result for the delay management problem with the additional delay objective function. The proof involves a different reduction from independent set, and generally needs more than three passenger transfers.

Lemma 4. *For any $\epsilon > 0$, it is NP-hard to approximate the binary decision delay management problem with the objective of minimizing the additional delay within a factor $\frac{15}{14} - \epsilon$.*

Proof. It is easy to see that the problem is in NP, as the weighted delay of the paths induced by a delay policy can be computed in polynomial time, and the size of the delay policy is polynomial as well. We first construct an alternative reduction from Independent Set [GJ79, Problem GT20] to show that the problem is NP-hard, and then use this reduction to prove the inapproximability result.

The idea of the alternative reduction is the following. For each vertex in the independent set instance, we insert a so-called vertex-path that is maintained if the vertex is in the independent set. As in the proof

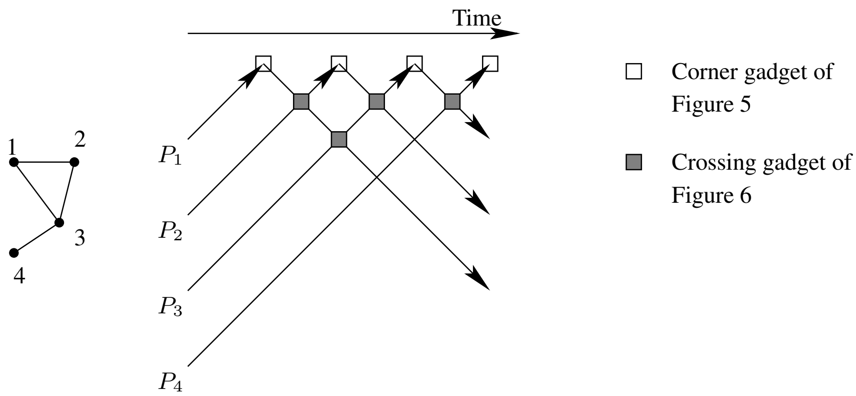

Figure 4: An example of the reduction. At the left, the independent set graph, at the right, the resulting delay management network. Paths are drawn as arrows to indicate their direction.

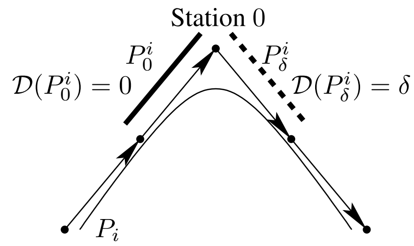

Figure 5: The corner gadget, enforcing a vertex-path to be dropped if already delayed before station 0, and delaying it after 0. Thick paths are M-paths, dashed paths have source delay.

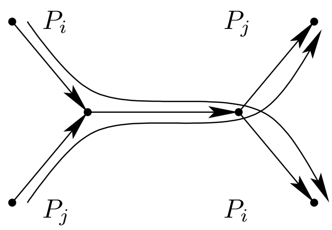

Figure 6: The crossing gadget, enforcing two paths whose vertices share an edge in G to use a common train in the delay management problem. Here, paths are drawn as arrows to indicate their direction.

of Theorem 1, two adjacent vertices yield two vertex-paths that share an edge in such a way that at most one of the two paths can be maintained in any optimal delay policy. There is a one-to-one correspondence between the vertices of an independent set and the maintained paths in the resulting delay management instance.

More precisely, given an instance $G = (V, E), K \in \mathbb{N}, |V| = n, |E| = m$, of Independent Set, we construct an instance of the delay management problem $(G', \mathcal{P}, \mathcal{D}, w, T)$, $G' = (V', E')$, as follows. Set $\delta = 1, T = 2, d = 2n - K$, and consider an arbitrary ordering of V from 1 to n. For presentation purposes, the construction is embedded in the plane, with 'time' on the horizontal axis, and 'space' on the vertical axis. For each vertex $i \in V$, we construct a unit weight source punctual vertex-path P_i. We specify the edges of the path below. The path P_i starts at location i at time zero, reaches location 0 at time i, and ends its journey at location $n - i$ at scheduled time n. To perform this journey, the path P_i needs to connect between trains several times. An example of the embedding is sketched in Figure 4.

At location 0 we introduce a so-called corner gadget, illustrated in Figure 5. Each path P_i arrives at station 0 with an edge it shares exclusively with a source punctual M-path P_0^i. The interaction of P_0^i with the network is limited to this edge, and to the path P_i. Similarly, P_i leaves station 0 through an edge shared exclusively with a source delayed M-path P_δ^i, which interacts with the network solely through this edge. By choosing M big enough, delaying P_0^i is more expensive than dropping P_i. In this way, P_i is forced to be dropped if it reaches the edge in common with P_0^i delayed. Similarly, P_i is forced to be delayed if it reaches the edge common with P_δ^i on time.

Two adjacent vertices $i, j \in V, i < j$, cannot both be in the independent set. To enforce this, the paths P_i and P_j cross exactly once on one common edge, in a so-called crossing gadget as shown in Figure 6. In the embedding, the crossing gadget is placed after P_i's corner gadget but before P_j's corner gadget, see Figure 4. Thus, given the decision to maintain path P_i, the path P_j must be dropped, as it reaches its corner gadget with a delay. Conversely, given the decision to maintain the path P_j, the path P_i must be dropped, as it reaches the crossing gadget with a delay. Hence, the two paths cannot be maintained concurrently if there is an edge $(i, j) \in E$. Note that the paths are disjoint if $(i, j) \notin E$.

Set $M = 2n$. Delaying one of the M-paths causes a delay of $2n \cdot \delta = 2n$, which is equal to the delay caused by dropping all unit weight vertex-paths. However, at least one vertex-path can always be maintained by neither delaying nor dropping any M-paths. In the worst case, all remaining vertex-paths must be dropped, which yields an additional weighted delay of $2n-1$. Thus, delaying or dropping M-paths cannot lead to an optimal policy. Again, the size of the instance remains polynomial when each M-path is replaced by M unit weight paths, as $M = 2n$.

The graph G has an independent set of size at least K if and only if the constructed unweighted delay management instance has a delay policy inducing at most $2n - K$ additional weighted delay (which corresponds to a total delay of at most $2n - K + nM\delta$). The reduction is polynomial, since each vertex induces $1 + 2M = 1 + 4n$ paths and $O(m)$ edges, and the instance can be built in polynomial time.

Finally, we show the inapproximability result for the additional weighted delay objective. As the vertices of the maintained vertex-paths form an independent set in G, the vertices of the dropped vertex-paths form a vertex cover in G. Hence, G has a vertex cover of size at most c if and only if there exists a delay policy with additional delay at most $n+c$. As shown in [Hås01], it is NP-hard to distinguish graphs having a vertex cover of size $\leq (\frac{6}{8} + \epsilon)n$ from those having a vertex cover of size $\geq (\frac{7}{8} - \epsilon)n$. This provides an inapproximability result of $\frac{7}{6} - \bar{\epsilon}$.

Our objective has an additive offset n. Hence, distinguishing the above delay management instances with additional delay $n + \frac{6}{8}n = \frac{14}{8}n$ from those with additional delay $n + \frac{7}{8}n = \frac{15}{8}n$, is equivalent to distinguishing between the corresponding sizes of a vertex cover. The ratio between these values proves the statement. □

3.4 Polynomially Solvable Cases

Several special cases of the delay management problem can be solved by reduction to a minimum directed cut problem [GGJ+04]. This section extends these results for the case in which no delay policy is allowed to drop source punctual paths. As dropping source punctual paths is in some sense unfair, this restricted case may very well be reasonable from a practical point of view.

A minimum directed s-t-cut is a partition of the vertex set into two disjoint sets $S, \bar{S}$, with $s \in S, t \in \bar{S}$, such that the sum of the cost of the edges traversing the cut from S to $\bar{S}$ is minimal. We construct a new graph $G' = (V', E', c)$, with a cost function $c : E \to \mathbb{N}$. A minimum directed cut in G' with respect to c corresponds to an optimal delay policy on $(G, \mathcal{P}, \mathcal{D}, w, T)$.

We map the trains E to vertices in G', and add two new vertices s and t. The idea of the reduction is that trains in S wait, whereas trains in $\bar{S}$ depart on time. For each source punctual passenger path we introduce infinite weight edges between every two subsequent trains the path uses. Further, for each such path we introduce a new vertex and infinite weight edges from each train used by the path to the new vertex, such that it will be in S if the path is delayed. An appropriately weighted edge connected to $t \in \bar{S}$ accounts for the delay occurring if the path is delayed. For each source delayed path, we also introduce a new vertex. This vertex is connected to all trains used by the path with infinite weight edges, such that the vertex is in $\bar{S}$ if one of these trains is on-time. An appropriately weighted edge connects $s \in S$ with this vertex, accounting for the dropping costs. An additional weighted edge (s, t) accounts for the delay of the source delayed path.

More precisely, given an instance $(G, \mathcal{P}, \mathcal{D}, w, T)$, we build a new graph $G' = (V', E', c)$, $c : E' \to \mathbb{N}$. Let $V' = E \cup \{t, s\} \cup \mathcal{P}$. For each source punctual path $P \in \mathcal{P}, P = \{f_0, f_1, \ldots f_l\}, f_i \in E, \mathcal{D}(P) = 0$, we introduce edges $(f_i, f_{i+1}), i \in \{0, \ldots, l-1\}$ each with weight $c(f_i, f_{i+1}) = \infty$. Further, let $v_P \in V'$ be the vertex corresponding to this path. We introduce edges $(f_i, v_P), i \in \{0, \ldots, l\}$ with weight $c(f_i, v_P) = \infty$, and an edge (v_P, t) with weight $c(v_P, t) = \delta \cdot w(P)$. For each source delayed path $P \in \mathcal{P}, P = \{f_0, \ldots, f_l\}, f_i \in E, \mathcal{D}(P) = \delta$, let v_P be the vertex corresponding to the considered path. We introduce edges $(v_P, f_i), i \in \{0, \ldots, l\}$ with weight $c(v_P, f_i) = \infty$, an edge (s, v_P) with weight $c(s, v_P) = (T - \delta) \cdot w(P)$ and finally an edge (s, t) with weight $c(s, t) = \delta \cdot w(P)$.

Lemma 5. *Given that source punctual passenger paths cannot be dropped, the minimum total passenger delay for* $(G, \mathcal{D}, \mathcal{P}, w, T)$ *is equal to the cost of the minimum directed* s*-*t*-cut* $[S, \bar{S}]$ *in* G'*. In such an optimal delay policy all trains corresponding to vertices in* S *wait and all trains corresponding to vertices in* $\bar{S}$ *depart on-time.*

Proof. It is clear that at least one non-infinite cut exists, since all source delayed paths can be dropped by setting $S = \{s\}$. Next, we show that source punctual paths can never be dropped. Since finite weight edges appear only from and to vertices v_P, and between s and t, the infinite weight edges ensure the desired consistency. Assume that the path $P = \{f_0, \ldots, f_l\} \in \mathcal{P}, \mathcal{D}(P) = 0$, is delayed. Then, there exists an $f_i \in P \cap S$. Because of the infinite weight edges $(f_i, f_{i+1}), \ldots (f_{l-1}, f_l)$, no such edge can traverse the cut in a minimum directed cut. Hence, the path will not be dropped. Notice that such paths can be delayed, as the infinite weight edges traverse the cut backwards from $\bar{S}$ to S and are hence not counted in the objective.

We show that the cost of each non-infinite weight cut is equal to the delay occurring if all trains in S wait, and all trains in $\bar{S}$ depart on-time. Each source punctual path $P = \{f_0, \ldots, f_l\}$, has $v_P \in S$ if one $f_i \in S$, i.e., if the path is delayed. If this were not the case, at least one infinite weight edge would traverse the cut. Since $v_P \in S$, the edge (v_P, t) traverses the cut, adding the delay costs for path P. If the path is not delayed, then $v_P \in \bar{S}$, and accordingly no edge related to P traverses the cut in any direction. For each source delayed path $P = \{f_0, \ldots, f_l\}, f_i \in E, \mathcal{D}(P) = \delta$, per construction (s, t) traverses the cut, adding the delay costs of the path to the cut's size. Further, assume one of the trains used by P departs on-time, and hence the path is dropped. Then, the vertex v_P must be in $\bar{S}$, as an infinite weight edge would traverse the cut otherwise. Correspondingly, the edge (s, v_P) is in the cut, increasing the cut's costs for this path to $T \cdot w(P)$. If the path is not dropped, then $v_P \in S$, and no other edge traverses the cut. $\square$

Now, if the source punctual paths are short enough that they cannot be dropped, the above construction works in general. This observation yields the corollary below.

Corollary 6. *The delay management problem with an unrestricted delay policy can be solved by reduction to a minimum cut problem if each source punctual path uses one train only.*

Moreover, a slightly different construction than above for the source punctual paths yields the following theorem.

Theorem 7. *The delay management problem with an unrestricted delay policy can be solved by reduction to a minimum cut problem if each source punctual path transfers at most twice.*

Proof. For the source delayed paths, we apply the same construction as in the proof of Lemma 5. For the source punctual paths, we use a different construction which is similar to the one in [GGJ$^+$04], see Figure 7.

For this gadget, we analyze what happens for a path $P = \{e_1, e_2, e_3\}$ of length three with weight $w(P)$. First, due to the infinite weights of the edges (e_i, v_P), the vertex v_P is in S whenever at least one of the vertices $\{e_1, e_2, e_3\}$ is in S. When v_P is in S, the edge (v_P, t) contributes $\delta \cdot w(P)$ to the cut cost, which reflects the weighted delay of the passenger path P.

Second, P can only be dropped when changing from e_1 to e_2 or when changing from e_2 to e_3. In both cases, only one edge with weight $(T - \delta) \cdot w(P)$ can be in the cut, increasing the total cost of the cut to $T \cdot w(P)$. This is exactly the cost of missing a connection.

$\square$

4 Delay Management with Slack Times

In this section, we analyze the case where trains can have some slack time. A train e having slack time $\mathcal{S}(e)$ is able to catch up $\mathcal{S}(e)$ time units on its delay. As stated earlier, we do not allow trains to catch up more time than they are delayed, implying that they can never arrive early. Also for this case, the never-meet-property allows for a polynomial-time algorithm [Sch03].

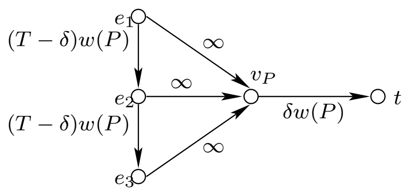

Figure 7: Gadget for a source punctual path $P = \{e_1, e_2, e_3\}$

4.1 Proof of Hardness

In Section 3.1, we showed that the delay management problem is NP-complete on general networks. Here, we show that by including slack times the delay management problem becomes NP-complete already with passenger paths transferring twice. In contrast, the delay management problem without slack times is still polynomially solvable on a line network [GGJ+04] .

We reduce from Maximum Directed Acyclic Cut. As the hardness proof of Maximum Directed Cut in [PY91] does not create an acyclic graph, we first show that the Maximum Directed Acyclic Cut problem is NP-complete.

Definition: Maximum Directed Acyclic Cut
Instance: Directed acyclic unweighted graph $G = (V, E)$, $K \in \mathbb{N}$.
Question: Does a partition of V exist into two disjoint sets $V_1, V_2, V = V_1 \cup V_2$, such that the number of edges traversing the partition from V_1 to V_2 is greater than or equal to K?

Lemma 8. *Maximum Directed Acyclic Cut is NP-complete.*

Proof. Clearly, the problem is in NP. We prove it to be NP-hard by reduction from Maximum Unweighted Directed Cut [GJ79, Problem ND16]: given a directed graph $G = (V, E), |V| = n, |E| = m$ and a positive integer $K \in N$, is there a partition of V into two disjoint sets $V_1, V_2, V = V_1 \cup V_2$, such that the number of edges traversing the cut from V_1 to V_2 is at least K?

We first build a maximum directed acyclic cut instance $G' = (V', E')$ using edge weights c' as follows. For each vertex $v_i \in V$, we build a gadget of five vertices, $\{v_i^1, v_i^2, v_i^3, v_i^4, v_i^5\}$, connected by four edges $(v_i^j, v_i^{j+1}), j \in \{1, \ldots, 4\}$, with weight $c'(v_i^j, v_i^{j+1}) = m$. It is clear that at most two non-consecutive edges of each gadget can traverse the cut. By setting their weights to m we enforce that two of these edges actually do traverse the cut. For each edge $e = (v_i, v_j) \in E$, we insert the edge (v_i^2, v_j^4) in E' with weight $c'(v_i^2, v_j^4) = 1$.

The reduction is polynomial in space and time: we have $5n$ vertices and $4n + m$ edges, and the graph can be constructed efficiently. The graph G has a maximum cut V_1, V_2 of size K if and only if G' has a maximum cut V_1', V_2' of size $2nm + K$.

The crucial observation for the reduction's correctness is that a gadget cannot have both $v_i^2 \in V_1'$ and $v_i^4 \in V_2'$, and at the same time contribute $2m$ from gadget-internal edges.

Finally, the above reduction also works for unweighted graphs G'. In that case we introduce, for each gadget, m parallel paths of length two between even-numbered vertices instead of the edges of weight m, multiplying the odd-numbered vertices. Still, the cut consistently separates the vertices as above. As we introduce $4m$ edges for each vertex in G, the construction remains polynomial. □

In the following, we show that fairly restricted versions of the delay management problem with slack times are already NP-complete.

Definition: Decision delay management problem with slack times.
Instance: A delay management instance $(G, \mathcal{S}, \mathcal{P}, \mathcal{D}, w, T)$, $d \in \mathbb{N}$.
Question: Does a delay policy exist, such that the total passenger delay does not exceed d?

Theorem 9. *The decision delay management problem with slack times is NP-complete with binary delays, binary slack times, unweighted passenger paths, and passengers transferring at most twice.*

Proof. The proof is by reduction from Maximum Directed Acyclic Cut. It is clear that the problem is in NP, as the delay of each path can be efficiently computed from a delay policy.

Given a maximum directed acyclic cut instance $G = (V, E)$, we build a delay management problem $(G', \mathcal{P}, \mathcal{D}, w, \mathcal{S}, T)$, with $G' = (V', E')$, as follows. For every $v \in V$, we introduce an edge $f_v \in E'$ without slack. For each edge $e = (u, v) \in E$, we introduce an edge g_e from f_u to f_v having slack time equal to δ. Further, for each edge $e = (u, v) \in E$, we introduce two paths, the path $P_e = (f_u, g_e, f_v)$ with source delay δ with unit weight, and the source punctual path $P_e^u = \{f_u\}$ with weight 3. More precisely, the latter weighted path can be replaced by three parallel paths of unit weight. Note that each outgoing edge $(u, v) \in E$ induces one P_e^u path on f_u.

We set $\delta = 1$ and $T = 4$, and ask for a delay policy inducing a total delay of $d = mT - K\delta = 4m - K$. There is a direct correspondence of a delay policy in G' to a cut in G: if f_u waits, $u \in V_1$, otherwise $u \in V_2$. It remains to argue that we have a cut of size at least K if and only if there is a delay policy with at most d total delay. To this end, it is sufficient to analyze the delay caused by the two paths P_e and P_e^u for the different policies. If f_u does not wait, P_e is dropped and P_e^u is on time. So, independent of f_v, these two paths together contribute T to the objective. If both f_u and f_v wait, the paths contribute $4\delta = T$ to the objective, as both paths arrive with a delay. Only if f_u waits and f_v departs as scheduled, the two paths contribute 3δ to the objective. Now, G has a maximum directed cut of size K if and only if $(G, \mathcal{S}, \mathcal{P}, \mathcal{D}, w, T)$ has a delay policy causing $4m - K = d$ delay. Using the described correspondence between a cut in G and a delay policy in G', for every edge e of G there is a contribution of 3 units to the total delay if e crosses the cut, and otherwise of 4 units. □

In contrast to Lemma 5, no source punctual paths are dropped in the above construction. Note that the reduction can be adapted to any $T = k\delta$ by introducing $k - 1$ paths P_e^i per edge e instead of three. The special case $k = 1$ is also feasible, but it is unclear how this should be interpreted. Furthermore, dynamic path choices do not influence the construction of Theorem 9, since the first and the last edge of the paths P_e cannot be changed. This observation allows us to simplify the network topology even further:

Corollary 10. *The decision delay management problem with slack times is NP-complete with binary delays, binary slack times, and unweighted passenger paths, even if the network forms a line.*

Proof. Since G is acyclic, we can assume that the numbering of the vertices induces a topological ordering, i.e., for every edge $(u_i, u_j) \in E$ we have $i < j$. Now replace the edges g_e by edges g_i connecting f_{u_i} with $f_{u_{i+1}}$, with unit slack time. Additionally, each path P_e for $e = (u, v)$ now follows the line from f_u to f_v. Since all g_i have slack time equal to one, P_e can only be dropped at its first edge. □

Finally, the structure of the created instance has the following natural interpretation. The trains without slack times stand for real trains, the edges with slack times stand for transfers at the stations. Naturally, the first and last activity of a passenger path are on a real train. Given that the slack time for transferring at stations is large enough, there is no propagation of the delays.

In general, the above proof of Corollary 10 yields paths with an arbitrary number of transfers, as opposed to the proof of Theorem 2.

Below, we show that the delay management problem with slack times is already NP-complete on a series-parallel network where passengers transfer at most twice.

Corollary 11. *The decision delay management problem with slack times is NP-complete on a series-parallel network if passenger paths transfer twice, with unit path weights, binary delays, and binary slack times.*

Proof. The reduction is from Maximum Cut on a directed graph $G = (V, E), |V| = n, |E| = m$. Combining Lemma 8 and the construction of Theorem 9, we arrive at a network G'.

Note that w.l.o.g., each node v_i^1 from Lemma 8 can be placed in U_1, and all v_i^5 in U_2. This leads to some simplification of the paths in G', schematically depicted in Figure 8. More precisely, the path P_v^4 would originally continue to one more edge without slack, which is superfluous. Similarly the path P_v^1 originally starts at an earlier edge, but is never dropped there.

As in the proof of Theorem 2, the network G' can be made series-parallel by contracting nodes with the same functionality, see Figure 9.

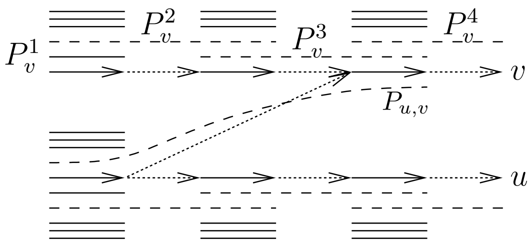

Figure 8: The delay management instance with slack times resulting from an edge (u, v) in the Maximum Cut instance G. Directed edges represent trains. Dotted edges represent trains with slack time one, plain edges have no slack time. Undirected lines represent paths. Solid paths are source punctual, dashed paths have source delay 1.

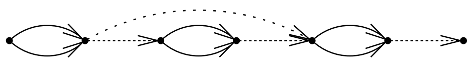

Figure 9: The contraction of the corresponding nodes (here shown for the network above) leads to three bundles of parallel edges on edges with no slack time.

Thus, the network consists of only 7 nodes, and bundles of parallel edges. Note that each bundle of edges with slack time can be contracted to a single edge with slack time, since every train with slack time can wait, and still arrive at its destination punctually. Each path now uses its original edges without slack times, and the contracted edges with slack time. Further, each path still interacts with the other paths on the same edges without slack time as before the contraction. □

The proof of Corollary 11 might suggest that re-routing passenger paths simplifies the problem. This is not the case, since the route of each path can be made unique by the techniques sketched in Section 3.2. In this case, we only identify the second and the fifth node. Note that the edge with slack between these two nodes can be removed as well, since the paths $P_{u,v}$ can be rerouted through other existing edges between the two nodes. Except for this latter type of path, all other paths have a unique route, and each route for $P_{u,v}$ induces the same costs.

4.2 Polynomially Solvable Cases

Although the general setting on the line is NP-hard, some variants of the delay management problem with slack times can be solved efficiently by simple strategies. Below we describe two such variants.

Let G be a graph that forms a line. Contrary to the models analyzed so far, we consider a single train traveling on the line with intermediate stops. This implies that a passenger path need not connect to other trains, once it has entered the train. Hence, a path can either be dropped before boarding the train, or it reaches its destination, possibly with some delay.

First, assume that all paths $P \in \mathcal{P}$ end at the terminal station of the considered train. This can be interpreted as passengers traveling to the city center on an urban rail line. We refer to this model as *all passengers to a unique destination on a single train*.

Theorem 12. *The delay management problem with slack times and all passengers to a unique destination on a single train can be solved in polynomial time.*

Proof. This problem can be solved by the following pedal-to-the-metal strategy. The driver a priori fixes a target delay at the terminal stop, exhausts all slack times, and drives at maximum velocity to achieve that target delay.

Given a target arrival delay δ, the delay policy π corresponding to the pedal-to-the-metal strategy is computed in a backward fashion, starting from the arrival station. Let δ_i be the delay at station i. The delay of the train at station $i-1$ is computed as $\delta_{i-1} = \delta_i + \mathcal{S}((i-1, i))$. Any optimal policy must use all the available slack, since using less slack could only result in dropping more passenger paths. Thus, the policy π is optimal for the arrival delay δ.

To see which values of the target arrival delay δ are relevant, consider $\delta = 0$. Let $l(\delta)$ be the minimum time by which a passenger path missed the train aiming at a target arrival delay δ. If the train had waited $l(\delta)$ longer, thus targeting for an arrival delay $\delta + l(\delta)$, only this path would additionally be maintained. Hence, we analyze the target arrival delay $\delta + l(\delta)$ next. Since the values $\delta \in (\delta_i, \delta_i + l(\delta_i))$ result in dropping the same paths as for δ_i but increase the arrival delay, they need not be considered. This procedure can then be iterated. As at most $|\{P \in \mathcal{P} : \mathcal{D}(P) > 0\}|$ paths can lead to a different arrival delay, only polynomially many solutions are evaluated, and we can pick the best one. □

All Passengers to Unique Destination through Single Links on a Rooted Tree
The above delay policy can be extended to the case where single trains operate between the stations, the graph is a rooted in-tree, and all passengers travel to the root of the tree. The passengers must thus connect to a new train at each intermediate station on their trip to the root node.

We point out that it makes no sense to drop a connection in the middle of a passenger path. By doing this, we would drop all paths starting before that connection. But then, we may as well have the preceding trains wait less, and maintain the connections to the paths which can board the train with these delays. Since we then drop less connections than before, or in extreme cases the same connections, this delay policy is optimal on a line. It is also optimal for a tree, as the argument holds for every incoming edge of a node. Hence, the pedal-to-the-metal strategy for all passengers to a unique destination on a train can be applied to this problem as well.

When G is a general tree, this kind of strategy also works if all paths start at a common stop. The same holds if all passengers travel in the same direction through a common station.

5 Conclusions and Future Work

We resolved the complexity status of the delay management problem by proving it to be NP-complete, even for quite restricted problem variants. Still, the research on delay management is only at the beginning, and many other computational aspects are yet to be explored. For example, good approximation algorithms provide a challenging direction for further theoretical research.

On the other hand, there is an obvious need to solve practical instances. Even if our polynomially solvable cases are quite simplified, they still indicate parameters that are crucial for the problem's complexity. A challenging task is to develop algorithms and heuristics for real-world problems. More specifically, decomposition methods or branching schemes could be based on passenger paths with more than three transfers, or on sub-networks with a line or tree topology.

Acknowledgement We thank Peter Widmayer for many fruitful discussions.

References

[ADGT99] B. Adenso-Diaz, M. Gonzalez, and P. Gonzalez Torre. On-line timetable re-scheduling in regional train services. *Transportation Research-B*, 33/6:387–398, 1999.

[Bod93] H. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[BS04] C. Biederbick and L. Suhl. Improving the quality of railway dispatching tasks via agent-based simulation. In *Computers in Railways IX*, pages 785–795. WIT Press, 2004.

[GGJ+04] M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Algorithm Theory - Proceedings SWAT 2004*, pages 199–211. Springer-Verlag LNCS 3111, 2004.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[GJPW04] M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. On-line delay management on a single line. In *Proc. Algorithmic Methods and Models for Optimization of Railways (ATMOS)*. Springer-Verlag LNCS, 2004. To appear.

[Hås01] J. Håstad. Some optimal inapproximability results. *Journal of ACM*, 48:798–859, 2001.

[HHW99] D. Heimburger, A. Herzenberg, and N. Wilson. Using simple simulation models in the operational analysis of rail transit lines: A case study of the MBTA's red line. *Transportation Research Record*, 1677:21–30, 1999.

[HK81] C. Hendrickson and G. Kocur. Schedule delay and departure time decisions in a deterministic model. *Transportation Science*, 15:62–77, 1981.

[Meg04] C. Megyeri. Bicriterial delay management. Konstanzer Schriften in Mathematik und Informatik 198, University of Konstanz, 2004.

[OW99] S. O'Dell and N. Wilson. Optimal real-time control strategies for rail transit operations during disruptions. In N. Wilson, editor, *Lecture Notes in Economics and Mathematical Systems: Computer-Aided Transit Scheduling*, volume 471. Springer, 1999.

[Pol74] S. Poljak. A note on the stable sets and coloring of graphs. *Comment. Math. Univ. Carolin.*, 15:307–309, 1974.

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[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.

[Sch01] A. Schöbel. A model for the delay management problem based on mixed-integer-programming. In Christos Zaroliagis, editor, *Electronic Notes in Theoretical Computer Science*, volume 50. Elsevier, 2001.

[Sch03] A. Schöbel. *Customer-oriented optimization in public transportation*. Habilitation Thesis, University of Kaiserslautern, 2003. 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.