Railway Delay Management: Exploring its Algorithmic Complexity

Michael Gatto, Björn Glaus, Riko Jacob, Leon Peeters, Peter Widmayer

Institute of Theoretical Computer Science, ETH Zurich
{gattom,leon.peeters,rjacob,widmayer}@inf.ethz.ch,glauss@student.ethz.ch

Abstract. We consider delay management in railway systems. Given delayed trains, we want to find a waiting policy for the connecting trains minimizing the weighted total passenger delay. If there is a single delayed train and passengers transfer at most twice along fixed routes, or if the railway network has a tree structure, the problem can be solved by reduction to min-cut problems. For delayed passenger flows on a railway network with a path structure, the problem can be solved to optimality by dynamic programming. If passengers are allowed to adapt their route to the waiting policy, the decision problem is strongly $\mathcal{NP}$-complete.

1 Introduction

In recent years, there has been an increasing focus on delays in railway systems, and many railway companies focus on reducing delays in an effort to increase customer satisfaction. The possibility that one train gets delayed is always present. In order to allow passengers to transfer from a delayed feeder train, it can be beneficial to deliberately delay a connecting train. This can save delayed passengers from having to wait for the next train. In this case, some of the feeder's delay is knocked on to the connecting train, and all passengers already in the connecting train also face a delay. Moreover, these passengers may miss their connection if they wish to transfer in a subsequent station.

In general, it might thus be beneficial to propagate the delays in the network, and one should decide on a set of waiting trains. Even today, such waiting policy decisions are usually taken by a human dispatcher. Ideally, a modern railway decision support system should enable a dispatcher to easily evaluate the impact of his decisions, or even propose a good waiting policy. This naturally leads to the algorithmic problem of determining an optimal waiting policy that minimizes the overall passenger delay.

Although delay management problems have been studied for some years, not much is known about the computational complexity of minimizing total passenger delay in general models. In this paper, we analyze restricted variants of the event-activity network model presented in [Nac98], which was analyzed in [Sch02]. In this graph representation, vertices represent arrival and departure events at stations. Directed edges represent driving and waiting activities for trains, as well as transfer activities for passengers. Trips of passenger are modeled as paths in the network. Each path has a weight representing its importance.

One of the models considered in [Sch02] is based on a bi-criterial objective function. The goal is to simultaneously minimize the perturbation of the timetable and the total weight of paths that miss a connection. This version of the problem is known to be weakly $\mathcal{NP}$-hard [Sch02]. The total delay can be efficiently minimized if the "never-meet-property" holds, which basically forbids two delayed vehicles meeting at a station. The general model can be solved to optimality through an ILP-based branch-and-bound algorithm [Sch01,Sch02]. Other theoretical models consider on-line versions with unknown delays at one bus stop [APW02] or the influence of buffer times for delays with exponential distribution for one transfer [Gov98,Gov99]. A fair amount of work was done using simulations [Man01,HHW99,OW99]. However, these last studies are less related to this paper.

As the complexity of the general event-activity network is still unknown, we focus on a restricted setting, as formalized below. The basic element of our model is a direct link between two stations and each such link is operated by a single train. Further, we assume that the original timetable is tight, so there is no possibility for a train to catch up on a delay. In the same spirit we assume that all the transfers at one station happen instantaneously. Hence there is only one amount of delay necessary. Obviously this is a fairly strong restriction of the original model. Nevertheless, we are convinced that the key combinatorial structure remains. As soon as the decision on which passengers to drop are taken, it is easy to produce a modified timetable that minimizes the remaining delays. The general model as well as this model are easily solvable as soon as we know how many passengers use a certain transfer. The problem seems to be that dropping passengers somewhere has a significant effect on these numbers throughout the network. Our model singles out this particular phenomenon. As we cannot even analyze this restricted model in its entirety, it appears that we did not strip away all of the complication of the event-activity network.

For simplicity, we include the externally caused delay of passengers in the objective function. Clearly, such delays cannot be optimized, and their contribution in the objective function is known a priori. As long as we focus on optimal solutions, this offset has no impact on the complexity. Note, however, that this offset can make a considerable difference for approximation ratios and competitive analyses.

We analyze three different cases of our model: (i) a single delayed train in the network with passenger following a predefined path; (ii) origin-destination paths for passenger flows which can have primary unit delays on a railway corridor; (iii) a single delayed train and origin-destination pairs for the passengers, who can adapt their route according to the delays in the network. The primary path delays in the second model may seem unusual, but they should be interpreted as passengers arriving at a station on a delayed train and wishing to transfer. Note that an instance of (ii) can be mapped to an instance of (i) by introducing some additional connections and infinite weight passenger paths. However, this usually obscures the graph structure of the original problem. Hence, we analyze these two models separately.

In this paper, we present the following results. We show that (i) can be solved efficiently if passengers transfer at most twice, or if the railway network has a tree structure. We further describe some extensions to more complicated models. For model (ii) we present a polynomial time algorithm for railway corridors. Finally, we show that problem (iii) is strongly $\mathcal{NP}$-hard.

2 General Problem Statement

This section describes the characteristics common to our three models. The specific characteristics of each model are described in the corresponding sections.

Let $G = (V, E)$ be a directed acyclic graph with the vertices in V are topologically sorted. Each vertex of the graph represents a station. An edge $e = (u, v) \in E$ represents a direct link from station u to station v operated by a single train. At every station $v \in V$ the outgoing edges (v, u) represent the possible connections for all incoming edges (w, v). Thus, a directed path within G corresponds to a journey a passenger can undertake by transferring to other trains at each intermediate station.

We distinguish between externally caused primary delays, and secondary delays, which are introduced by the waiting policy. We study the case in which all primary delays are identical and of size δ. All transfers at a station take place instantaneously, so the passengers of a delayed train miss their connection unless it waits for them. Observe that a connecting train must wait for the entire delay δ of the delayed feeder train in order to maintain a connection. In this case, all transfers to the connecting train are maintained, and the entire delay δ propagates. Additionally, we assume that a delayed train cannot catch up on its delay.

All trains are operated according to a periodic timetable with period T, and we assume that delays do not propagate to the next period. So, if a person misses a train, she continues her journey with the next trains traveling along the same route. Since there are no further disturbances in the next period, her arrival is thus delayed by T time units. Our analyses below do not depend on the fact that all passengers face the same period T. Indeed, our results can be extended to models where passengers missing a connection incur a delay depending on their route, and on the station where they miss their connection.

We consider passenger flows in the railway network as a set of paths $\mathcal{P}$ in G. For each passenger path $P \in \mathcal{P}$, we are given a source vertex $s(P)$ and a target vertex $t(P)$. For a pair of vertices $u, v \in V$, we allow multiple paths in $\mathcal{P}$. Each path $P \in \mathcal{P}$ is defined as the ordered set of edges leading from $s(P)$ to $t(P)$. Further, each path $P \in \mathcal{P}$ has a weight $w(P)$, which represents the number of passengers, or the path's importance in a more abstract sense.

Our objective is to minimize the total weighted delay for all passenger paths in the network, given the primary delays δ. A path $P \in \mathcal{P}$ contributes to the objective function as follows: 0 if P arrives as scheduled, $\delta \cdot w(P)$ if P arrives with a delay of δ, and $T \cdot w(P)$ if P misses a connection. We refer to paths P

with zero delay as on time paths, to those facing a delay of $\delta \cdot w(P)$ as delayed paths, and to paths facing a delay of $T \cdot w(P)$ as dropped paths.

3 Minimum Cut Reductions for Single Primary Delays

In this section, we consider the previously described model for the case in which there is a single train e_0 with primary delay. The passenger paths $\mathcal{P}$ are assumed as given. We stress that these paths are fixed, and will be followed by the passengers whatever delays will be introduced by the delay policy. We assume that the vertex u of the primary delayed edge $e_0 = (u, v)$ has in-degree zero, as every instance can be transformed to an equivalent instance having this quality. Let $\Delta \subseteq E, e_0 \in \Delta$ be the set of delayed trains, $\Omega = E \setminus \Delta$ the on-time trains.

We define the single source delay management problem: given an instance $(G, e_0, \mathcal{P}, w, \delta, T)$, find a set $\Delta \subseteq E$ of waiting trains minimizing the total weighted delay on the network.

In the following, we show how we can transform this restricted problem to a minimum cut problem. We introduce the method for passenger paths with at most one transfer and then extend it to two transfers. Thus, the single source delay management problem can be efficiently solved if passengers switch trains at most twice, i.e. if they use at most three different trains.

The key idea is to build a new graph N in which every $s-t$-cut $[S, \bar{S}]$ represents the delay policy $\Delta = S, \Omega = \bar{S}$. To do this, the trains (edges) of the original graph G are mapped to vertices in the graph N; forward and backward edges are added to N between vertices e and f if in the original graph passengers can connect from e to f. More edges are added, and the weights are defined such that the cost of a cut will correspond to the total delay occurring if trains in S are delayed and trains in $\bar{S}$ depart on time.

Given the train network $(G = (V, E), e_0, \mathcal{P}, w, \delta, T)$, the equivalent directed minimum $s-t$-cut network $N = (H = (U, F), s, t, c), \quad s, t \in U, \quad c : F \to \mathbb{N}$ is built as follows. Set $U = E \cup \{t\}, \quad s = e_0$; let $F = F_1 \cup F_2$, where $F_1 = \{(e, f), (f, e) | e, f \in E, e = (u, v), f = (v, w)\}$ and $F_2 = \{(e, t) | e \in E\}$. Let $P = \{e, f\}, e, f \in E$ be an arbitrary passenger path of length 2, from source station $s(P)$ to target station $t(P)$. The passengers in P change train exactly once. Let $\mathcal{P}(e_i)$ be the set of paths using edge e_i. We define the edge costs in N as (see Figure 1 for an example):

$$c(e, f) = \begin{cases} (T - \delta) \cdot w(P) & \text{if } P = \{e, f\} \in \mathcal{P}; \\ \delta \cdot w(P) & \text{if } P = \{f, e\} \in \mathcal{P}; \\ \delta \cdot \left(\sum_{P \in \mathcal{P}(e)} w(P) \right) & \forall e \in U, f = t \end{cases}$$

Let $G' = (V', E')$ be a directed graph, $c : E' \mapsto \mathbb{N}$ a weight function and let $\{s', t'\} \subset V'$. A directed cut is a partition of V' into two sets S' and $\bar{S}' = V' \setminus S'$, such that $s' \in S', t' \in \bar{S}'$. Letting $E'_c = \{(e, f) \in E' : e \in S', f \in \bar{S}'\}$, the cost of the cut is defined as $C(S', \bar{S}') = \sum_{(e,f) \in E_c} c(e, f)$.

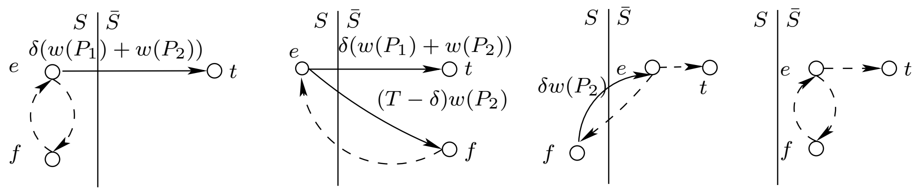

Fig. 1. The four possible cuts for two connecting trains with paths $P_1 = \{e\}$ and $P_2 = \{e, f\}$ of weight $w(P_1)$ and $w(P_2)$, respectively.

Δ	Ω	Total weighted delay
$\{e, f\}$	$\emptyset$	$\delta \cdot (w(P_1) + w(P_2))$
$\{e\}$	$\{f\}$	$\delta \cdot w(P_1) + T \cdot w(P_2)$
$\{f\}$	$\{e\}$	$\delta \cdot w(P_2)$
$\emptyset$	$\{e, f\}$	0

Fig. 2. The four possible delay policies for two connecting trains with paths $P_1 = \{e\}$ and $P_2 = \{e, f\}$, with weight $w(P_1)$ and $w(P_2)$, respectively.

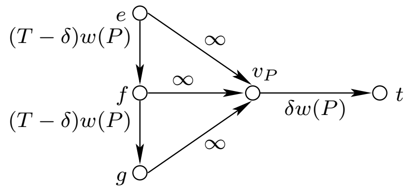

Fig. 3. Reduction for a path $P = \{e, f, g\}$

Lemma 1. *The cost of a minimum directed $s-t$-cut $[S, \bar{S}]$ in N is equal to the minimum total weighted delay in G for the delay policy $\Delta = S, \Omega = \bar{S} \setminus \{t\}$, given that passengers connect at most once.*

The proof is by analysis of the cases shown in Figure 1 and Figure 2. Details can be found in [GGJ$^+$04].

This approach can be extended to passenger paths changing trains twice. The key idea is to build an additional structure for each path P of length three. Basically, we use the same construction as in the previous reduction for accounting the $(T - \delta)$ weight which delayed paths experience when they are dropped. For paths using three trains at most one such edge can traverse the cut. As for the δ delay, we cannot maintain the previous reduction, as two edges weighted with $\delta \cdot w(P)$ could traverse the cut. This problem can be solved by introducing an additional vertex for each path. The weights of the edges connecting it will be such that it will be in S as soon as one vertex of the path is in S. Finally, we can add an edge connecting the vertex to t, with weight $\delta \cdot w(P)$: this edge will be in the cut as soon as P is delayed, providing the correct weight for being delayed. So, we extend the previous reduction for paths P composed by three edges, implying a double transfer, as follows (see Figure 3). For each such path $P = \{e_1, e_2, e_3\}$, we add a vertex v_P. The vertices $e_i \in P$ are connected to v_P through edges (e_i, v_P) of weight $c(e_i, v_P) = \infty$. Further, we add edges $(e_1, e_2), (e_2, e_3),\ w(e_1, e_2) = w(e_2, e_3) = (T - \delta) \cdot w(P)$, and an edge (v_p, t) of weight $w(v_p, t) = \delta \cdot w(P)$.

Lemma 2. *The cost of every directed $s-t$-cut $[S, \bar{S}]$ in N is equivalent to the total delay on the network G when applying the policy $\Delta = S^-, \Omega = \bar{S}^-$, where*

$S^- = S \setminus \{v_P : P \in \mathcal{P}, |P| = 3\}, \bar{S}^- = \bar{S} \setminus (\{t\} \cup \{v_P : P \in \mathcal{P}, |P| = 3\})$, *given that passengers transfer at most twice. Hence, the minimum directed $s-t$-cut in N is equivalent to the minimum delay strategy on G, given that passengers transfer at most twice.*

Again, the proof is by case analysis, refer to [GGJ+04] for details.

Theorem 1. *The minimum delay policy for $(G, e_0, \mathcal{P}, w, \delta, T)$ where paths do not transfer more than twice can be found efficiently by reduction to a polynomial size directed minimum $s-t$-cut problem.*

Proof. When building the graph N, we introduce a vertex for each train and for each path of length three, i.e. $O(|E| + |\mathcal{P}|)$ vertices. Similarly, we introduce one edge (e, t) for each train and $O(1)$ edges for each path, i.e. $O(|E| + |\mathcal{P}|)$ edges in total. A minimum directed cut on a graph with n vertices and m edges can be found in $O(nm \log n)$ time[ST83]. Hence, we need $O((|E| + |\mathcal{P}|)^2 \log(|E| + |\mathcal{P}|))$ to find a minimum delay policy.

4 Minimum Cut Reductions for Special Cases

Next, we show how the min-cut approach can be extended to tree-like networks with arbitrary paths as well as to trains with intermediate stops with passengers transferring at most once.

The proposed reduction can be adapted straight forwardly to paths with arbitrary length, given that the graph G is an out-tree. [Sch02] showed that the "never-meet-property" applies to out trees, hence the delay management problem can be solved efficiently. Our reduction is another method to solve this special network structure.

The minimum cut approach can also be extended to train networks where trains have intermediate stops, given that the passengers transfer at most once. We assume the passenger paths are given as in single source delay management problem. The set $\mathcal{R}$ of trains with intermediate stops are represented as non-empty, edge disjoint paths, $R = \{f_1, \ldots, f_r\}, f_i \in E, 1 \leq i \leq r$. All edges in the graph are direct train connections. By restricting $|R| = 1, \forall R \in \mathcal{R}$, we get the single source delay management problem. Since we assume tight timetables, all subsequent edges in a train's path are delayed as soon as some edge delays.

Similarly to the previous reductions, we map the edges E to vertices U, add a vertex t to U and set $s = e_0$. For each train $R = \{f_1, f_2, \ldots, f_r\}$ we introduce edges $(f_i, f_{i+1}), 1 \leq i < r$ with weight $w(f_i, f_{i+1}) = \infty$. As soon as $f_i \in S$, such edges prevent the vertices $f_k, i < k \leq r$ from being in $\bar{S}$, providing the required consistency that delayed trains cannot catch up their delay. Hence, each train in $\mathcal{R}$ can become delayed, but it will never be on time again.

The edges and their weights are defined as follows. For all the paths $P = \{f_1, \ldots, f_l\}$, $f_i \in R$ using only train R, and which hence do not connect, we introduce the edges $(f_1, t), (f_l, f_1)$ with weights $w(f_1, t) = w(f_l, f_1) = \delta \cdot w(P)$. For the paths $P = \{f_1, \ldots, f_l, g_1, \ldots, g_m\}, f_i \in R_1, g_i \in R_2$, which use two trains and thus need to connect, we must extend the single train reduction above. As before, we introduce $(f_1, t), (f_l, f_1)$, $w(f_1, t) = w(f_l, f_1) = \delta \cdot w(P)$. These edges account for the delay that can be accumulated on the first train. Additionally, we introduce (f_l, g_1), $w(f_l, g_1) = (T - \delta) \cdot w(P)$; such edges account for the additional dropping delay passengers experience if the first train is delayed and the second departs on time. Finally, (g_m, f_l), $w(g_m, f_l) = \delta \cdot w(P)$ accounts for the delay passengers experience if they get delayed on the second train.

Lemma 3. *The cost of the minimum $s - t$-cut $[S, \bar{S}]$ on N equals the total weighted delay of the minimum delay waiting policy for $(G, e_0, \mathcal{P}, \mathcal{R}, w, \delta, T)$, given that passengers transfer at most once.*

For the proof and other details, we refer to [GGJ$^+$04].

5 Multiple Path Delays on a Railway Corridor

This section describes a model for multiple primary delays on a railway corridor. For this model, we present an enumeration tree, and show that equivalent subtrees of that enumeration tree can be pruned. This observation leads to a polynomial time algorithm for the minimum total weighted passenger delay objective. Finally, we show that the pruned enumeration tree algorithm boils down to a dynamic programming algorithm.

5.1 The model

The model in this section has two specific characteristics. First, we consider a corridor in the railway network, and deal with the primary delays that enter that corridor through delayed feeder trains. Since a corridor corresponds to a path in the railway network, this implies that G is a path. Second, each passenger path $P \in \mathcal{P}$ has a primary delay $\delta(P) \in \{0, 1\}$. Such a delayed passenger path should be interpreted as having arrived with some delayed feeder train. The primary path delay can take only two values, either one time unit or zero time units. However, our analysis below only uses the fact that all non-zero primary delays are identical. As before, this implies that a connecting train either departs as scheduled, or it waits for all delayed passengers.

In this case, we denote the ordered vertex set by $V = (v_1, \ldots, v_{m+1})$, with $v_1 < \ldots < v_{m+1}$, and the edge set by $E = (e_1, \ldots, e_m)$, with $e_i = (v_i, v_{i+1})$. The decision whether a connecting train waits for delayed passengers or not is modeled by the following wait-depart decision variable:

$$x_i = \begin{cases} 1 \text{ if train } e_i \text{ delays its departure from station } v_i \text{ by one time unit,} \\ 0 \text{ if train } e_i \text{ departs from station } v_i \text{ as scheduled.} \end{cases}$$

As before, the objective function of the model is to minimize the total weighted path arrival delay over all possible vectors $(x_1, \ldots, x_m) \in \{0, 1\}^m$.

5.2 Binary enumeration tree

This section describes an m–level binary enumeration tree H for the values of the decision variables x_i. In H, branching between the levels $i-1$ and i represents choosing a value for the variable x_i. A node at level i in H has a label $\langle x_1, \ldots, x_i \rangle$, where $x_1, \ldots, x_i$ are the decisions taken on the unique path from the root node to that node. The root node itself has the empty label $\langle\rangle$. So, the label $\langle x_1, \ldots, x_i \rangle$ immediately contains the partial solution at the node, and a leaf node $\langle x_1, \ldots, x_m \rangle$ at level m represents a solution to the model.

At node $\langle x_1, \ldots, x_i \rangle$, wait-depart decisions have been taken for the trains $e_1, \ldots, e_i$. We keep track of the impact of these decisions through the following functions of the nodes (we omit the node label brackets here to improve readability):

$$
\begin{aligned}
A(x_1, \ldots, x_i) &= \left\{ P \in \mathcal{P} \,\middle|\, s(P) \leq v_i < t(P), P \text{ not dropped by } x_1, \ldots, x_i \right\} \\
D(x_1, \ldots, x_i) &= \sum_{\substack{P \in \mathcal{P} \\ \text{dropped by } x_1, \ldots, x_i}} T \cdot w(P) + \sum_{\substack{P \in \mathcal{P}: t(P) \leq v_i \\ \text{delayed by } x_1, \ldots, x_i}} w(P) \\
D_m(x_1, \ldots, x_m) &= D(x_1, \ldots, x_m) + \sum_{P \in A(x_1, \ldots, x_m)} w(P) \cdot x_m
\end{aligned}
$$

The set of active passenger paths $A(x_1, \ldots, x_i)$ contains all paths P that are traveling in train e_i, given the decisions at node $\langle x_1, \ldots, x_i \rangle$. In a sense complementary, $D(x_1, \ldots, x_i)$ contains the already accumulated weighted delay caused so far by the decisions at node $\langle x_1, \ldots, x_i \rangle$. So, at any level i in H, each path $P \in \mathcal{P}$ with $s(P) \leq v_i$ is either contained in $A(x_1, \ldots, x_i)$, or its weighted arrival delay is accounted for in $D(x_1, \ldots, x_i)$. Note that $A(x_1, \ldots, x_i)$ contains paths P with $t(P) = v_{i+1}$, although the arrival delay for such paths is known when the decisions $x_1, \ldots, x_i$ have been taken. In particular, this means that the active path set $A(x_1, \ldots, x_m)$ at a leaf node may be non-empty. Therefore, $D_m(x_1, \ldots, x_m)$ accounts for the weighted arrival delay at v_{m+1} of all passenger paths P on train e_m, given the decisions at node $\langle x_1, \ldots, x_m \rangle$. An optimal solution to our model is then represented by a leaf node $\langle x_1, \ldots, x_m \rangle^*$ that attains a minimum value $D_m(x_1, \ldots, x_m)$.

For the root node $\langle\rangle$, we set $D() = 0, A() = \emptyset$. Below, we specify the initialization for the child nodes $\langle 0 \rangle$ and $\langle 1 \rangle$ of the root node $\langle\rangle$. Next, we describe a general child node $\langle x_1, \ldots, x_i, x_{i+1} \rangle$ with parent node $\langle x_1, \ldots, x_i \rangle$. Since the values of $A(x_1, \ldots, x_{i+1})$ and $D(x_1, \ldots, x_{i+1})$ depend on the parent's values and on the values of the decisions x_{i+1} and x_i, we distinguish between the four possible combinations for a child node $\langle \ldots, x_i, x_{i+1} \rangle$.

Initialization of $\langle 0 \rangle$ and $\langle 1 \rangle$

$$
A(0) = \{P \in \mathcal{P} | s(P) = v_1, \delta(P) = 0\} \qquad A(1) = \{P \in \mathcal{P} | s(P) = v_1\}
$$

$$
D(0) = \sum_{\substack{P \in \mathcal{P}: \\ s(P) = v_1, \delta(P) = 1}} T \cdot w(P) \qquad D(1) = 0
$$

$\langle \ldots, 0, 0 \rangle$ nodes

$$A(x_1, \ldots, x_{i+1}) = A(x_1, \ldots, x_i) \setminus \{P \in \mathcal{P} | t(P) = v_{i+1}\} \\ \cup \{P \in \mathcal{P} | s(P) = v_{i+1}, \delta(P) = 0\}$$

$$D(x_1, \ldots, x_{i+1}) = D(x_1, \ldots, x_i) + \sum_{\substack{P \in \mathcal{P}: \\ s(P) = v_{i+1}, \delta(P) = 1}} T \cdot w(P)$$

$\langle \ldots, 0, 1 \rangle$ nodes

$$A(x_1, \ldots, x_{i+1}) = A(x_1, \ldots, x_i) \setminus \{P \in \mathcal{P} | t(P) = v_{i+1}\} \cup \{P \in \mathcal{P} | s(P) = v_{i+1}\}$$

$$D(x_1, \ldots, x_{i+1}) = D(x_1, \ldots, x_i)$$

$\langle \ldots, 1, 0 \rangle$ nodes

$$A(x_1, \ldots, x_{i+1}) = \{P \in \mathcal{P} | s(P) = v_{i+1}, \delta(P) = 0\}$$

$$D(x_1, \ldots, x_{i+1}) = D(x_1, \ldots, x_i) + \sum_{\substack{P \in A(x_1, \ldots, x_i): \\ t(P) > v_{i+1}}} T \cdot w(P) + \sum_{\substack{P \in \mathcal{P}: \\ s(P) = v_{i+1}, \delta(P) = 1}} T \cdot w(P) + \sum_{\substack{P \in A(x_1, \ldots, x_i): \\ t(P) = v_{i+1}}} w(P)$$

$\langle \ldots, 1, 1 \rangle$ nodes

$$A(x_1, \ldots, x_{i+1}) = A(x_1, \ldots, x_i) \setminus \{P \in \mathcal{P} | t(P) = v_{i+1}\} \cup \{P \in \mathcal{P} | s(P) = v_{i+1}\}$$

$$D(x_1, \ldots, x_{i+1}) = D(x_1, \ldots, x_i) + \sum_{\substack{P \in A(x_1, \ldots, x_i): \\ t(P) = v_{i+1}}} w(P)$$

As an example, and since it is a special case, we briefly discuss the case of a $\langle \ldots, 1, 0 \rangle$ node. Because $x_i = 1$ and $x_{i+1} = 0$, all paths in $A(x_1, \ldots, x_i)$ not ending in v_{i+1} cannot transfer to train e_{i+1} and are dropped, facing a weighted arrival delay of $T \cdot w(P)$. Train e_{i+1} is also missed by all paths $P \in \mathcal{P}$ starting at v_{i+1} with primary delay $\delta(P) = 1$, so these paths also face a weighted arrival delay of $T \cdot w(P)$. Further, since $x_i = 1$, all paths $P \in A(x_1, \ldots, x_i)$ ending at v_{i+1} arrive with a weighted arrival delay of $w(P)$. Finally, the only paths P that do depart with train e_{i+1} are those starting at v_{i+1} and having $\delta(P) = 0$.

5.3 Pruning equivalent subtrees

A $\langle \ldots, 1, 0 \rangle$ enumeration tree node implies that all passengers wanting to transfer from train e_i to train e_{i+1} will miss their connection, so none of the paths in $A(x_1, \ldots, x_i)$ enters $A(x_1, \ldots, x_i, x_{i+1})$. Therefore, the subtree rooted at node $\langle x_1, \ldots, x_{i+1} \rangle$ is in a sense independent of the decisions $x_1, \ldots, x_i$ taken before. The Lemmas below show that this independence allows to prune the enumeration tree significantly.

Lemma 4. *For an enumeration subtree rooted at node $\langle x_1, \ldots, x_{i+1} \rangle$, let*

$$D_{\langle x_1, \ldots, x_i \rangle}(x_{i+1}, \ldots, x_{i+k}) := D(x_1, \ldots, x_{i+k}) - D(x_1, \ldots, x_i)$$

be the accumulated weighted delay in the subtree node $\langle x_{i+1}, \ldots, x_{i+k} \rangle$. *Any two enumeration subtrees rooted at* $\langle x_1, \ldots, x_i, x_{i+1} \rangle$ *and* $\langle x'_1, \ldots, x'_i, x'_{i+1} \rangle$ *with* $x_i = x'_i = 1$ *and* $x_{i+1} = x'_{i+1} = 0$ *are equivalent in the sense that*

$$D_{\langle x_1, \ldots, x_i \rangle}(x_{i+1}, \ldots, x_{i+k}) = D_{\langle x'_1, \ldots, x'_i \rangle}(x_{i+1}, \ldots, x_{i+k}), \text{ for } k = 1, \ldots m - i.$$

Proof. We use the shorthand notation $\langle \ldots, 1, 0 \rangle$ for some node with a label $\langle x_1, \ldots, x_{i-1}, 1, 0 \rangle$. From the construction of $\langle \ldots, 1, 0 \rangle$ nodes, it is clear that

$$A(\ldots, 1, 0) = \{P \in \mathcal{P} | s(P) = v_{i+1}, \delta(P) = 0\}.$$

So, $A(\ldots, 1, 0)$ is the same for each node $\langle \ldots, 1, 0 \rangle$. Therefore, both the enumeration subtrees rooted at $\langle x_1, \ldots, x_{i+1} \rangle$ and $\langle x'_1, \ldots, x'_{i+1} \rangle$ start with the same path set $A(\ldots, 1, 0)$, and will thus have accumulate the same weighted delay until subtree node $\langle x_{i+1}, \ldots, x_{i+k} \rangle$.

Lemma 5. *Of all subtrees rooted at* $\langle x_1, \ldots, x_{i+1} \rangle$, *with* $x_i = 1, x_{i+1} = 0$, *it suffices to explore the single subtree with root*

$$\langle x_1, \ldots, x_{i+1} \rangle^* = \underset{\langle x_1, \ldots, x_{i+1} \rangle}{\mathit{argmin}} \left\{ D(x_1, \ldots, x_{i+1}) \middle| \, x_i = 1, x_{i+1} = 0 \right\}$$

Proof. Lemma 4, with $k = m - i$, implies that the minimum of $D(x_1, \ldots, x_{i+1}, x_{i+2}, \ldots, x_m)$ and $D(x'_1, \ldots, x'_{i+1}, x_{i+2}, \ldots, x_m)$ is determined by the minimum value of $D(x_1, \ldots, x_{i+1})$ and $D(x'_1, \ldots, x'_{i+1})$.

5.4 Analysis of the algorithm

Because of the pruning described in Lemma 5, the number of nodes in the enumeration tree can be reduced significantly from $O(2^m)$ to $O(m^3)$, as is stated by the following Lemma. Moreover, this leads to an overall worst case running time of $O(m^5)$ for the pruned enumeration tree algorithm.

Lemma 6. *The pruned enumeration tree has* $O(m^3)$ *nodes.*

Proof. In order to count the number of nodes in the enumeration tree, we define the following variables:

$N^0(i)$	The number of nodes $\langle x_1, \ldots, x_{i-1}, 0 \rangle$ at level i.
$N^1(i)$	The number of nodes $\langle x_1, \ldots, x_{i-1}, 1 \rangle$ at level i.

From the definition of the initialization phase, it follows that $N^0(1) = N^1(1) = 1$. At level $i+1$, a child node $\langle \ldots, 1 \rangle$ is created for every parent node $\langle \ldots, 0 \rangle$, and also for every parent node $\langle \ldots, 1 \rangle$. Therefore, $N^1(i+1) = N^0(i) + N^1(i)$. Further, for every parent node $\langle \ldots, 0 \rangle$, a child node $\langle \ldots, 0 \rangle$ is created. But, by Lemma 5, one single child node $\langle \ldots, 0 \rangle$ is created for all parent nodes $\langle \ldots, 1 \rangle$. This yields $N^0(i+1) = N^0(i) + 1$. Solving the recurrences

$$N^0(1) = 1 \qquad\qquad N^1(1) = 1$$
$$N^0(i+1) = N^0(i) + 1 \qquad\qquad N^1(i+1) = N^0(i) + N^1(i)$$

gives $N^0(i) = i$ and $N^1(i) = \frac{1}{2}i^2 - \frac{1}{2}i + 1$. So, the total number of nodes at level i is $O(m^2)$. With m levels, the pruned enumeration tree has $O(m^3)$ nodes in total.

Theorem 2. *The railway delay management problem on a corridor with multiple* $\{0, 1\}$ *primary passenger path delays can be solved in* $O(m^5)$ *time.*

Proof. The pruned enumeration tree has $O(m^3)$ nodes, and at each node at most $|\mathcal{P}| = O(m^2)$ paths have to be evaluated to compute the functions $A(.)$ and $D(.)$.

5.5 A dynamic programming view

This section shows that the pruned enumeration tree algorithm can also be written as a dynamic program. We use the function $z_{10}(i, k)$ that denotes the minimum value of all partial solutions where anything can have happened until train e_{k-2}, train e_{k-1} departs delayed, and all subsequent trains $e_k, \ldots, e_i$ depart as scheduled. Similarly, the function $z_{101}(i, j, k)$ denotes the minimum value of all partial solutions where anything can have happened until train e_{k-2}, train e_{k-1} departs delayed, the next trains $e_k, \ldots, e_{j-1}$ depart as scheduled, and all subsequent trains $e_j, \ldots, e_i$ depart delayed again. The subscripts for the functions z_{10} and z_{101} stand for the structure of the last significant events in the function's partial solutions.

The recursion formulas for $z_{10}(i, k)$ and $z_{101}(i, j, k)$ correspond uniquely to the four cases for the updates of the functions $D(x_1, \ldots, x_{i+1})$ for the enumeration tree node $\langle x_1, \ldots, x_{i+1}\rangle$. In particular, the case $i + 1 = k$ for $z_{10}(i + 1, k)$ corresponds to the pruning of the enumeration tree when only a single $\langle \ldots, 1, 0\rangle$ node is created at level $i + 1$. Because of these clear one-to-one correspondences, we do not describe the recursion formulas here, but refer to [GGJ+04] for a formal description.

6 Further Results

A careful analysis of the algorithm in Section 5 shows that it can be applied to an out-tree graph G. The dynamic program discloses that it can also be carried out backwards. Hence, it follows that the same approach can also be applied to in-trees.

Further, the dynamic programming algorithm can be extended for the case of K primary delay categories, that is, $\delta(P) \in \{\delta_1, \ldots, \delta_K\}$. However, this extension comes at the cost of a worst case running time of $m^{O(K)}$.

To our knowledge, no hardness result is yet known for the non-constrained single source delay management problem. Instead, we consider a model where passengers choose their route dynamically, according to the connections causing them the least delay. For this model, we have shown that:

Theorem 3. *The decision version of the delay management problem where passenger choose their route dynamically is* $\mathcal{NP}$*-complete.*

The reduction is from 3-SAT, and is omitted due to lack of space. It uses prominently that a route can and has to choose between alternatives, which makes it easy to ensure that only one of the literals is set to true, and also that every clause has at least one true literal. Refer to [GGJ$^+$04] for details.

7 Future Research

This paper discussed the algorithmic complexity of three variants of the event-activity model for railway delay management.

On the other hand, delay management is also an on-line problem. In such a setting, not the entire structure of the primary delays is known to the algorithm. On-line approaches are probably of particular interest to practical users. We have some preliminary results for on-line models on a path. Extensions to general graphs with multiple delays are of great interest, and decision policies for such models could be useful in real-life as well.

References

[APW02] L. Anderegg, P. Penna, and P. Widmayer. Online train disposition: to wait or not to wait? In Dorothea Wagner, editor, *Electronic Notes in Theoretical Computer Science*, volume 66. Elsevier, 2002.

[GGJ$^+$04] M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. Technical Report 441, ETH Zurich, Institute for Theoretical Computer Science, 2004.

[Gov98] R. Goverde. Optimal scheduling of connections in railway systems. Technical report, TRAIL, Delft, The Netherlands, 1998.

[Gov99] R. Goverde. Transfer stations and synchronization. Technical report, TRAIL, Delft, The Netherlands, 1999.

[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.

[Man01] S. Mansilla. Report on disposition of trains. Technical report, ETH Zurich, 2001.

[Nac98] K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables.* Habilitation Thesis, Braunschweig, Germany, 1998.

[OW99] S. O'Dell and N. Wilson. Optimal real-time control strategies for rail transit operations during disruptions. In *Computer-Aided Transit Scheduling*, pages 299–323. Lecture Notes in Economics and Math. Sys., Springer-Verlag, 1999.

[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.

[Sch02] A. Schöbel. *Customer-oriented optimization in public transportation.* Habilitation Thesis, Universität Kaiserslautern, 2002. To appear.

[ST83] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.