The Price of Robustness in Timetable Information*

Marc Goerigk[1], Martin Knoth[2], Matthias Müller-Hannemann[2], Marie Schmidt[1], and Anita Schöbel[1]

1 **Institut für Numerische und Angewandte Mathematik**
Georg-August Universität Göttingen, Germany
{m.goerigk,m.schmidt,schoebel}@math.uni-goettingen.de

2 **Institut für Informatik**
Martin-Luther-Universität Halle-Wittenberg, Germany
martin.knoth@student.uni-halle.de; muellerh@informatik.uni-halle.de

Abstract

In timetable information in public transport the goal is to search for a good passenger's path between an origin and a destination. Usually, the travel time and the number of transfers shall be minimized. In this paper, we consider *robust* timetable information, i.e. we want to identify a path which will bring the passenger to the planned destination even in the case of delays. The classic notion of strict robustness leads to the problem of identifying those changing activities which will never break in any of the expected delay scenarios. We show that this is in general a strongly NP-hard problem. Therefore, we propose a conservative heuristic which identifies a large subset of these robust changing activities in polynomial time by dynamic programming and so allows us to find strictly robust paths efficiently. We also transfer the notion of light robustness, originally introduced for timetabling, to timetable information. In computational experiments we then study the price of strict and light robustness: How much longer is the travel time of a robust path than of a shortest one according to the published schedule? Based on the schedule of high-speed trains within Germany of 2011, we quantitatively explore the trade-off between the level of guaranteed robustness and the increase in travel time. Strict robustness turns out to be too conservative, while light robustness is promising: a modest level of guarantees is achievable at a reasonable price for the majority of passengers.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; G.2.2 Graph Theory (Graph algorithms; Network problems)

Keywords and phrases Strict and Light Robustness; Delay Scenarios; Experimental Study

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.76

1 Introduction

Robust optimization takes into account that the input for optimization problems is uncertain to some extent. Many real-world applications share that there will be some kind of disturbance, e.g. input data changes, disruptions, delays or any other unforeseen event. To overcome such difficulties and make solutions applicable for real-world problems, researchers are working on various concepts of *robustness*. The goal of these concepts is not to find

* Partially supported by grants MU 1482/4-2 and SCHO 1140/3-1 within the DFG programme *Algorithm Engineering*. The authors wish to thank Deutsche Bahn AG for providing us with test data for scientific use.

the best solution to the (undisturbed) problem but to calculate a *robust* solution which is still 'good' in case of a disturbance. There are many promising concepts on how to define *robustness* of a solution. *Strict robustness* was introduced by Soyster [14] and significantly extended by Ben-Tal, El Ghaoui and Nemirovski, see [1, 2] and references therein. A solution to an optimization problem is called strictly robust if it is feasible for all possible scenarios. The idea of light robustness (see [7]) is to require a certain nominal quality and to look for a solution satisfying this quality and maximizing the robustness. In this paper we develop and evaluate the concepts of strict and light robustness for a real-life problem, namely the problem of determining a best path for traveling in a public transportation network.

Related work. The classical timetable information problem is usually modelled as a shortest path problem in either a time-expanded event-activity network or a time-dependent graph, see [12] for a survey. The "reliability" of a path has been considered as an additional search criterion within a multi-criteria timetable information system by Disser et al. [6]. Müller-Hannemann and Schnee [11] and Schnee [13] study timetable information in the presence of delays. They show that a massive stream of delay information and schedule changes can be efficiently incorporated into the search for optimal paths. The robust shortest path problem has found quite some attention in the literature [9]. Uncertainties are modeled by a set of known scenarios, where each scenario corresponds to a set of arc lengths (or weights). The *robust shortest path problem* is to find among all paths the one that minimizes the path length in the worst case over all scenarios. To the best of our knowledge, *robust timetable information* has not been studied before. State-of-the-art practical solutions allow to specify minimum transfer times, but usually they come without any guarantee of robustness.

Our contribution. The classical notion of strict robustness asks to find a solution which is feasible for any scenario. Translated to timetable information this leads to the problem of identifying those transfers which will never break subject to the specified set of delay scenarios. Surprisingly it turns out that already this problem of determining strictly robust changing activities is strongly NP-hard. Due to this hardness result, we use a conservative approximation, i.e. we forbid slightly more changing activities than necessary to guarantee strictly robust solutions. To this end, we compute the maximum amount of delay which can be accumulated for any arrival event. We succeed in developing a dynamic programming approach for this *delay accumulation problem* which runs in polynomial time for a realistic model of delay scenarios. We also transfer the concept of light robustness to timetable information and develop a solution approach. A light robust path is a path which may exceed the minimum travel time in the nominal scenario by not more than a certain specified amount but contains as few as possible changing activities which are not strictly robust under these restrictions. For both concepts we study the price of robustness, originally mentioned in [4]: How much longer is the travel time for a robust solution than for a shortest path according to the published schedule? We parametrize the set of considered delay scenarios by the maximum size and number of (large) delays which may occur. Each fixed parameter set can be interpreted as a level of robustness. In computational experiments with the schedule of high-speed trains within Germany of 2011, we explore the trade-off between the level of guaranteed robustness and the increase in travel time for both concepts. Strict robustness turns out to be too conservative, while light robustness is promising: a modest level of guarantees is achievable at a reasonable price for the majority of passengers.

Overview. In Section 2, we formally introduce event-activity networks as models for timetable information and introduce and discuss delay scenarios. To provide passengers with strictly robust timetable information, that is to find paths that are maintained in every

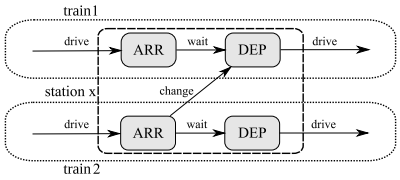

Figure 1 Detail of an event-activity network.

scenario, we need to identify the connections that cannot break. In Section 3, we study the computational complexity of finding these connections and prove NP-hardness of this problem. Due to this hardness result, we afterwards study the related delay accumulation problem which provides us with a subset of the connections that are always maintained. We derive a dynamic programming based algorithm to solve this problem. In this way we can solve the NP-hard problem of strictly robust timetable information heuristically in polynomial time. The concepts are extended to light robustness in Section 5. We present results of our computational study in Section 6 and finally conclude with remarks on future work. Due to the lack of space all proofs are omitted and can be found in the technical report [8].

2 Timetable information and delay models

Graph Model. In our paper we represent the timetable as an *event-activity network* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$. This is defined as follows: For every arrival and every departure of a train at a station we define an *event*. We also have two virtual events, to be described below. The events $\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep} \cup \mathcal{E}_{virt}$ are the nodes in the event-activity network. The edges are called *activities*. There are three groups of activities we consider: $\mathcal{A}_{drive}$ contains driving activities of a train between a departure and an arrival event. $\mathcal{A}_{wait}$ contains waiting of a train within a station (i.e. between an arrival and its following departure event), and $\mathcal{A}_{change}$ contains possible changing activities, i.e. transfers between the arrival of a train and the departure of another train (at the same station). Figure 1 shows an example for an event-activity network.

For each event $i \in \mathcal{E}$ the timetable provides a time $\pi_i \in \mathbb{N}$, usually in minutes. For each activity $a \in \mathcal{A}$ a length $l_a \in \mathbb{N}$ is given. This length represents the minimal duration the activity has, i.e. the minimal time that is required to drive between two stations (for driving activities). A feasible timetable hence satisfies that $\pi_j - \pi_i \geq l_a$ for all $a = (i, j) \in \mathcal{A}$. The *slack time* of an activity $a = (i, j) \in \mathcal{A}$ is defined as $s_a := \pi_j - \pi_i - l_a$. For our timetable information problem we furthermore have a request Req of a passenger. Such a request is specified by an origin station, a destination station and a time $t_{\text{request}} \in \mathbb{N}$ specifying when the passenger can start her journey. In order to model such a request in the event-activity network we add two virtual events, an origin event i_{org} and a destination event i_{dest} and the following set of virtual activities $\mathcal{A}_{virt}$: We connect the origin event to all events $i \in \mathcal{E}_{dep}$ starting at the origin station and having $\pi_i \geq t_{\text{request}}$, and we connect all events $j \in \mathcal{E}_{arr}$ belonging to the destination station and having $\pi_j \geq t_{\text{request}}$ to i_{dest}. If the passenger is interested in a path with earliest arrival time at her destination, we can solve this problem by determining a shortest path from i_{org} to i_{dest} with respect to the following weights

$$c_a := \begin{cases} \pi_j - \pi_i & \text{if } a = (i, j) \in \mathcal{A} \\ \pi_i - t_{\text{request}} & \text{if } a = (i_{org}, i) \in \mathcal{A}_{virt} \\ 0 & \text{if } a = (j, i_{dest}) \in \mathcal{A}_{virt}\,. \end{cases} \tag{1}$$

Summarizing, we denote the timetable information problem as $\mathbf{P}(\mathcal{E}, \mathcal{A}, \pi, \text{Req})$ and call it the *nominal problem*. The output is a shortest path P^* specified by its sequence of events, and the arrival time at i_{dest} denoted by $f(P)$.

Delay scenarios. If everything runs smoothly the passenger would be satisfied with such a shortest path. Unfortunately, delays are unavoidable. This is in particular annoying if a connection on such a path may be missed. The passenger hence may wish to have a reliable connection. To model the uncertainty we define a set of possible exogenous delays, called *source delays*, each of them increasing the lower bound of some activity duration l_a. Examples are obstacles on the tracks that have to be cleared before the train can pass or signalling problems. A scenario is hence given by a vector $d \in \mathbb{N}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|}$. In real world scenarios one often observes many small source delays, but only a few large ones (which have a direct or indirect effect on a passenger's path). Similar to Bertsimas and Sim [4], we take this into account and introduce a vector $\epsilon \in \mathbb{N}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|}$, specifying for each driving or waiting activity a an upper bound ϵ_a for a "small delay". Moreover, we assume that each source delay is bounded by $d_a^{\max}$ and that the total number of "large" source delays (i.e., those with $d_a > \epsilon_a$) is bounded by K for given values of $d_a^{\max}$ for all $a \in \mathcal{A}$ and an integer K. More precisely, the uncertainty set we consider is given as

$$\mathcal{U} := \mathcal{U}_\epsilon^K := \{d \in \mathbb{R}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|} : 0 \le d_a \le d_a^{\max} \text{ for all } a \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}, \\ |\{a \in A : d_a > \epsilon_a\}| \le K\}.$$

Delay propagation. When a scenario of source delays $d \in \mathcal{U}$ occurs, it spreads out through the network and results in new times $\pi_i(d)$ for the events $i \in \mathcal{E}$. The basic rule how delays spread is the following: If the start event of an activity $a = (i, j)$ is delayed, also its end event j will be delayed, where the delay can be reduced by the slack time s_a. I.e. we require $\pi(d) \ge \pi$ and

$$\pi_j(d) \ge \pi_i(d) + l_a + d_a \tag{2}$$

for all activities $a = (i, j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}$. For changing activities we have the following situation: If (2) holds for a changing activity we say that the connection is maintained. If (2) does not hold, the connection is broken and passengers cannot transfer between the corresponding events. This leads to a new set of changing activities which is denoted as $\mathcal{A}_{change}(d)$. In our paper we assume that the decision whether a connection should be maintained or not is specified by a fixed waiting time rule: Given a number $wt_a \in \mathbb{N}$ for any changing activity, the connection should be maintained if the departing train has to wait at most wt_a minutes compared to its original schedule.

Given these waiting time rules for a given delay scenario d we can propagate the delay through the network and thus calculate the corresponding adapted timetable according to the following propagation rule:

$$\pi_j(d) = \max\left\{\pi_j, \max_{i:(i,j)\in\mathcal{A};\ \pi_i(d)+l_{ij}\le\pi_j+wt_{ij}} \{\pi_i(d) + l_{ij} + d_{ij}\}\right\} \tag{3}$$

where we set $wt_a = \infty\ \forall a \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}$ and $d_a^{max} = 0\ \forall a \in \mathcal{A}_{change}$.

For the sake of tractability, this delay propagation model does not take microscopic conflicts like blocked tracks or platforms into account. However, these kind of secondary delays are captured by the small delays ϵ_a which may occur everywhere.

Timetable information under uncertainty. Both $\mathcal{A}$ and π are uncertain parameters for finding the required timetable information since they both depend on the set of source

delays d. We hence specify the timetable information problem under uncertainty as

$$\mathbf{P}(\mathcal{E}, \mathcal{A}_{drive}, \mathcal{A}_{wait}, \mathcal{A}_{change}(d), \pi(d), \text{Req}), \quad d \in \mathcal{U}.$$

3 Strictly robust timetable information

Applied to timetable information the concept of *strict robustness* requires that the path is "feasible" for all delay scenarios, i.e. that all its connections are maintained for any of the scenarios $d \in \mathcal{U}$. The set of strictly robust paths hence is $\mathrm{SR} = \{P :$ for all $d \in \mathcal{U}$ we have $P \cap \mathcal{A}_{change} \subseteq \mathcal{A}_{change}(d)\}$. In order to determine the set of robust paths, we have to analyze for every changing activity whether it is maintained in all scenarios:

(TT): Transfer-test. Given a changing activity $a = (i, j) \in \mathcal{A}_{change}$, does there exist a delay scenario $d \in \mathcal{U}$ such that a is not maintained?

The set of changing activities that are maintained for all scenarios $d \in \mathcal{U}$ is called the *set of strictly robust activities* and denoted by $\mathcal{A}^{\mathrm{SR}}$. Note that given $\mathcal{A}^{\mathrm{SR}}$, a robust path that has shortest travel time in the nominal case again can be easily computed using a shortest path algorithm in $\mathcal{N}^{SR} = (\mathcal{E}, \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \cup \mathcal{A}^{\mathrm{SR}} \cup \mathcal{A}_{virt})$.

▶ **Theorem 1.** *For the uncertainty set $\mathcal{U}^K_\epsilon$, (TT) is strongly NP-complete, even if $\epsilon_a = 0$ for all $a \in \mathcal{A}$.*

An intuitive explanation why transfer test is computationally hard is the following: whether a changing activity $a = (i, j)$ is maintained or not depends on the time values $\pi_i(d)$ *and* $\pi_j(d)$. Both values may or may not be influenced by the same source delay of some earlier event. So the core difficulty is to decide whether there is *no* delay scenario that simultaneously delays event i by a certain amount but does not delay event j by too much. We are not aware of any reasonable way to solve (TT) exactly.

Still, we can calculate a subset of the strictly robust connections using the following observation: Let $a = (i, j)$ be a changing activity. Then, if $\pi_i(d) \leq \pi_i + s_a + wt_a$ for all $d \in \mathcal{U}$, a is maintained for every delay scenario and thus $a \in \mathcal{A}^{SR}$. Thus the set of connections $\mathcal{A}^{acc}$ having this property is a subset of the strictly robust connections. Then, every path in the network $\mathcal{N}(\mathcal{A}^{acc}) = (\mathcal{E}, \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}^{acc} \cup \mathcal{A}_{virt})$ that contains only connections from $\mathcal{A}^{acc}$ is a strictly robust path. Note that to check the above-mentioned property, we only have to check whether the delay in i can exceed $s_a + wt_a$ or not. As we do not have to mind the consequences of the delay in j, this problem turns out to be much easier than (TT) as we will see in Section 4.

Slightly generalizing, this leads to the related problem:

(DA): Delay accumulation. Given an event $j^* \in \mathcal{E}$, an uncertainty set $\mathcal{U}$ and an integral number D, does there exist a delay scenario $d \in \mathcal{U}$ such that $\pi_{j^*}(d) = \pi_{j^*} + D$?

Let us explain how we can apply (DA). Consider again a changing activity $a = (i, j^*) \in \mathcal{A}_{change}$. If we solve (DA) for the event j^* and corresponding $D = s_a + wt_a + 1$, then the answer "no" proves that $a \in \mathcal{A}^{SR}$. This is sufficient because of the following monotonicity property: If there is a delay scenario which accumulates a delay of D at some event j^*, then it is also possible to generate every smaller delay at j^* (the latter is a consequence from Lemma 4 below). Hence, solving (DA) for every $a = (i, j^*) \in \mathcal{A}_{change}$ and corresponding $D = s_a + wt_a + 1$, we obtain a subset of the strictly robust connections $\mathcal{A}^{acc} \subset \mathcal{A}^{SR}$. Every path in the network $\mathcal{N}(\mathcal{A}^{acc})$ that contains only connections from $\mathcal{A}^{acc}$ is a strictly robust path. Thus given the network $\mathcal{N}(\mathcal{A}^{acc})$, we can solve the strictly robust timetable information

problem heuristically in polynomial time. A small observation that might be of theoretical interest is the following. (DA) is equivalent to (TT) if the underlying undirected graph of the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is acyclic or if $wt_a = 0$ holds for all $a \in \mathcal{A}_{change}$.

4 Efficiently solving delay accumulation

In the following we will show how problem (DA) can be solved in polynomial time. To this end we will derive properties of the delays that allow us to restrict to only a subset of delay scenarios when solving (DA). Due to this result we are able to develop Algorithm 1 that solves (DA) in polynomial time and can hence be used to determine $\mathcal{A}^{acc}$. As before, we will consider an event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ and delay scenarios d on $\mathcal{N}$. For an event $i \in \mathcal{E}$ and a delay scenario d we will denote by $d(i)$ the delay in node i, that is $d(i) = \pi_i(d) - \pi_i$, where $\pi_i(d)$ can be calculated successively for all nodes i using (3). Furthermore, by $\mathcal{N}(i)$ we will denote the events and activities of the network $\mathcal{N}$ from which a directed path to i exists in $\mathcal{N}$. We will refer to $\mathcal{N}(i)$ also as the "network preceding i".

The following lemma shows that it suffices to consider only the preceding network of a node i when calculating the possible delay at this node.

▶ **Lemma 2.** *Consider an event $j^* \in \mathcal{E}$ and a delay scenario d with $d(j^*) = D$. Then it holds that $d'(j^*) = d(j^*)$ for the delay scenario d' defined as*

$$d'_a := \begin{cases} d_a, & \text{if } a \in \mathcal{N}(j^*) \\ 0, & \text{if } a \in \mathcal{N} \setminus \mathcal{N}(j^*). \end{cases}$$

Note that if $d \in \mathcal{U}_\epsilon^K$ for a given K, also $d' \in \mathcal{U}_\epsilon^K$. Thus when trying to solve (DA) for a node j^* in $\mathcal{N}$, from now on we will restrict to delay scenarios having only delays in $\mathcal{N}(j^*)$.

In particular, we show in the following lemmata that we can assume that all delays lie on one single path toward j^*. This property is crucial for solving (DA).

▶ **Lemma 3.** *If for an event $j^* \in \mathcal{E}$ and a delay scenario d it holds that $d(j^*) > 0$, then there is at least one directed path P toward j^* such that for every $(i,j) \in P$*

$$\pi_j(d) = \pi_i(d) + l_{ij} + d_{ij} \text{ and} \tag{4}$$

$$wt_{ij} \geq \pi_i(d) - \pi_i - s_{ij}. \tag{5}$$

P contains at least one source delay.

We will call such a path P a "critical path for j^* and d". The next lemma shows that when we have a delay scenario causing a delay of D at a node j^*, we can also produce any amount of delay smaller than D at j^* by reducing the source delays in an appropriate way.

▶ **Lemma 4.** *Let d be a delay scenario and j^* a node with $d(j^*) = D$ for a $D \in \mathbb{N}$. Then there is a delay scenario d' with $d'_a \leq d_a$ for every $a \in \mathcal{A}$ and $d'(j^*) = D - 1$.*

The following Lemma 5 allows us to consider only delay scenarios where all delays lie on a critical path toward the considered node in (DA). In cases where we are interested in the delay in a specific node j^*, we will refer to delay scenarios where all occurring source delays lie on a critical path toward j^* as *path delay scenarios.*

▶ **Lemma 5.** *Let j^* be an event in $\mathcal{E}$ and d a delay scenario. Then there is a delay scenario d' with $d'_a \leq d_a \forall a \in \mathcal{A}$ and all arcs a with $d'_a > 0$ lying on a critical path P' toward j^* such that $d'(j^*) = d(j^*)$.*

Considering only path delay scenarios is the basic idea behind the dynamic algorithm. Note that when $d \in \mathcal{U}_\epsilon^K$ for given K and ϵ, also the path delay scenario d' constructed like in Lemma 5 is contained in $\mathcal{U}_\epsilon^K$. Thus every feasible delay scenario can be turned into a feasible path delay scenario causing the same delay in the regarded node. Consequently, in the following for solving the problem (DA) we will only look at path delay scenarios. Based on these observations, we can build a polynomial time dynamic-programming algorithm that for a given node j^* and a number D determines whether there is a path delay scenario that causes a delay of D at j^*. Starting with j^*, the algorithm goes backwards in the network and successively sets the node labels $d(j,k)$ which indicate how much delay is needed at node j to cause a delay of D at j^* under the assumption that at most $K-k$ large source delays on arcs succeeding j are set. Algorithm 1 summarizes this in pseudo code.

▶ **Theorem 6.** *For a given node j^*, an uncertainty set $\mathcal{U}_\epsilon^K$ and a number $D \in \mathbb{N}$, Algorithm 1 solves the problem (DA) in time $O(|\mathcal{A}|K)$:*
- If there is a delay scenario $d \in \mathcal{U}_\epsilon^K$ with $d(j^*) = D$, Algorithm 1 returns d.
- Otherwise, Algorithm 1 returns "No".

The set $\mathcal{A}^{acc}$ can now be obtained by using Algorithm 1 for every $a \in \mathcal{A}_{change}$ with $D := s_a + wt_a + 1$. The total complexity to do so is therefore $O(|\mathcal{A}||\mathcal{A}_{change}|K)$.

5 Light robust timetable information

Allowing only strictly robust solutions will often lead to paths with very long travel time that will probably not be accepted by the passengers. A promising alternative is light robustness. In our setting this means that the output for the passenger should be a path with reasonable length, that is, its length should not exceed the length of a nominal optimal path by too much. Among all solutions satisfying this criterion one looks for the "most robust" one, which we define as the one with the fewest number of unreliable transfers, i.e. such not contained in $\mathcal{A}^{\text{SR}}$. If additional information like probabilities for the unreliable transfers is given, weights can be introduced to differ between the grade of unreliability for these arcs.

For the robust timetable information problem we hence allow that the path gets longer in order to make it more robust: Let $f^* := f(P^*)$ denote the length of a shortest path for a request $\text{Req} = (u, v, t_{\text{request}})$ in the undisturbed scenario, and B a parameter bounding the allowed increase in travel time.

(Light-robust-path) Given a network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ a timetable π, a request Req consisting of an origin u, a destination v and a time t_{request}, and the set of strictly robust changing activities $\mathcal{A}^{\text{SR}}$, find a path P with length smaller or equal to $f^* + B$ that contains as few as possible changing activities not contained in $\mathcal{A}^{\text{SR}}$.

Given the set of strictly robust changing activities $\mathcal{A}^{\text{SR}}$ as defined in Section 3, we can find such a path using a shortest path algorithm minimizing the number of changing activities classified as being not strictly robust in an event-activity network where we exclude all events that take place later than $f^* + B$. This leads to the following lemma:

▶ **Lemma 7.** *Given the set of strictly robust connections $\mathcal{A}^{\text{SR}}$, (Light-robust-path) can be solved in polynomial time.*

Note that we assumed in the problem formulation of (Light-robust-path) that the set of strictly robust activities $\mathcal{A}^{\text{SR}}$ is given. As we have seen in Theorem 1, determining the set $\mathcal{A}^{\text{SR}}$ is strongly NP-hard in general. For finding a heuristic solution we can again consider the subset $\mathcal{A}^{acc}$ instead of $\mathcal{A}^{\text{SR}}$.

Algorithm 1 (Delay accumulation)

Require: Event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ with $\mathcal{A}$ topologically sorted (backwards), uncertainty set $\mathcal{U}_\epsilon^K$, event j^*, number D

Ensure: A delay scenario $d \in \mathcal{U}_\epsilon^K$ that causes a delay of at least D in j^* if that is possible. "No" otherwise.

Set $d(j,k) = \infty$, $succ(j,k) = \emptyset$ for all $k = 1, \ldots, K$, $j \in \mathcal{E}$.
$d(j^*, K) = D$
for $a \in \mathcal{A}$, topologically sorted backwards **do**
  Let $(i,j) = a$.
  for $k = K, K-1, \ldots, 1$ **do**
    if $a \in \mathcal{A}_{drive} \cup \mathcal{A}_{wait}$ **then**
      if $d(i,k) > \min\{d(j,k) + s_{ij} - \epsilon_{ij}, d(j,k+1) + s_{ij} - d_{ij}^{max}\}$ **then**
        $d(i,k) = \min\{d(j,k) + s_{ij} - \epsilon_{ij}, d(j,k+1) + s_{ij} - d_{ij}^{max}\}$,
        set $succ(i,k) := (j,k)$ or $succ(i,k) := (j,k+1)$ respectively.
      end if
    else if $a \in \mathcal{A}_{change}$ and $d(j,k) < wt_{ij}$ and $d(i,k) > d(j,k) + s_{ij}$ **then**
      $d(i,k) = d(j,k) + s_{ij}$ and $succ(i,k) := (j,k)$
    end if
    if $d(i,k) \leq 0$ **then**
      For $(j,l) = succ(i,k)$ set $d_{ij} := d(j,l)$ and set $(i',k') := (j,l)$.
      while $succ(i',k') \neq \emptyset$ **do**
        Set $(j,l) = succ(i',k')$
        if $l < k'$ **then**
          Set $d_{i'j} := d_{i'j}^{max}$
        else if $(i',j) \in \mathcal{A}_{drive} \cup \mathcal{A}_{wait}$ **then**
          Set $d_{i'j} = \epsilon_{ij}$.
        end if
        Set $(i',k') := succ(j,l)$
      end while
      Stop and **return** d.
    end if
  end for
end for
return "No".

Compared to the approach of strictly robust timetable information, light robust paths are not necessarily maintained under disruptions. But taken into account that passengers do not only wish to have a guaranteed travel route, but are willing to sacrifice some robustness for shorter travel times, this trade-off may be beneficial.

6 Empirical evaluation

Test instances and delay scenarios. Our computational study is based on the German train schedule of January 25-26, 2011, restricted to high-speed trains of the train categories intercity express ICE, intercity IC, and eurocity EC. Our event-activity network includes 771 trains, and 36588 events. We generated transfer arcs between pairs of trains at the same station, if the departing train is scheduled to depart not later than 120 minutes after the arrival time of the feeding train. Note that this gives us an implicit bound of 120 minutes for

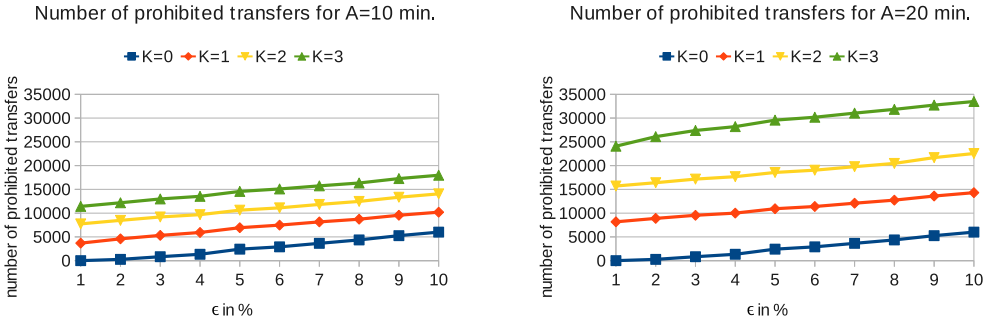

Figure 2 The number of transfer arcs which are infeasible according to delay accumulation for different parameter sets of the delay scenarios.

the maximum delay that robust paths can compensate for. However, we believe that this is sufficient for any reasonable strategy of robust pre-trip timetable information in practice. This leads to 51385 changing activities in our model. We applied the following standard waiting rule: Trains wait for each other at most 3 minutes. Passenger path requests have been generated by randomly chosen origins and destinations. Start times are chosen randomly in the interval of the first 12 hours of the day. To avoid trivial requests, we included only those requests for which the distance between start and destination is at least $150km$ and which require in the nominal scenario at least one transfer. In our experiments, we consider the scenario set $\mathcal{U}_\epsilon^K$. Our artificial delay scenarios are characterized by three parameters, ε, K, and A (with 80 different parameter settings in total):

- The parameter ε controls the maximal size of "small delays" which can occur in our model on every arc. This parameter has been varied between $\{0.01l_a, 0.02l_a, \ldots, 0.1l_a\}$, i.e., small delays are chosen as a fraction of the nominal length l_a of waiting and driving arcs.
- The second parameter K specifies the maximum number of "large delays" which may occur on some path. We assume that a passenger will be affected only by a small number of such "large delays", therefore we have K varied among $\{0, 1, 2, 3\}$.
- Finally, our third parameter A specifies the size of a maximal "large delay" if it occurs. Here we add the constant A to the maximal small delay of the arc. In our experiments, we used $A \in \{10, 20\}$ (in minutes).

Each parameter set can be interpreted as defining a certain "level of guaranteed reliability": Strict robust timetable information will deliver only paths for which all changing activities are immune against all delay scenarios described by this parameter set. Hence, the larger we choose these parameters, the stronger guarantees we obtain.

Test environment. All experiments were run on a PC (Intel(R) Xeon(R), 2.93GHz, 4MB cache, 47GB main memory under ubuntu linux version 10.10). Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.4.3 and compile option -O3.

Experiment 1 — strictly robust transfer arcs. In our first experiment, we want to study how many transfer arcs which exist in the nominal scenario are not strictly robust? And how does the number of prohibited transfer arcs depend on the parameters of the delay scenario? Given an overall number of 51385 changing activities, we observe that a considerable fraction becomes infeasible with increasing size of the delay parameters, see Figure 2. To determine

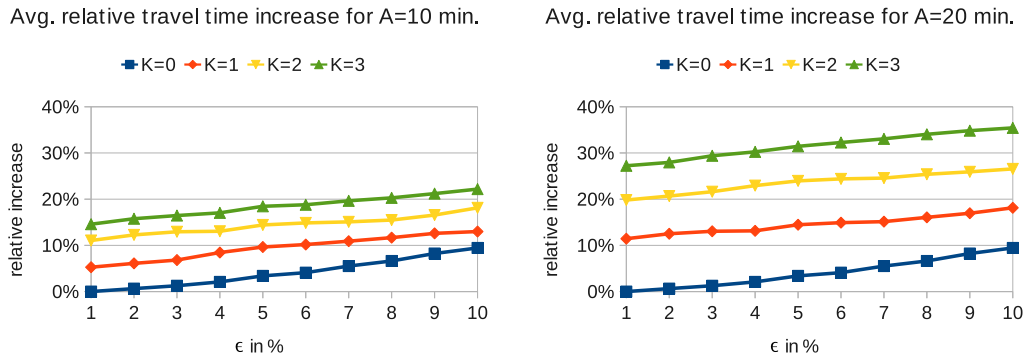

Figure 3 The average increase of travel time (in %) for quickest robust paths over optimal paths in the nominal scenario for different parameter sets of the delay scenarios.

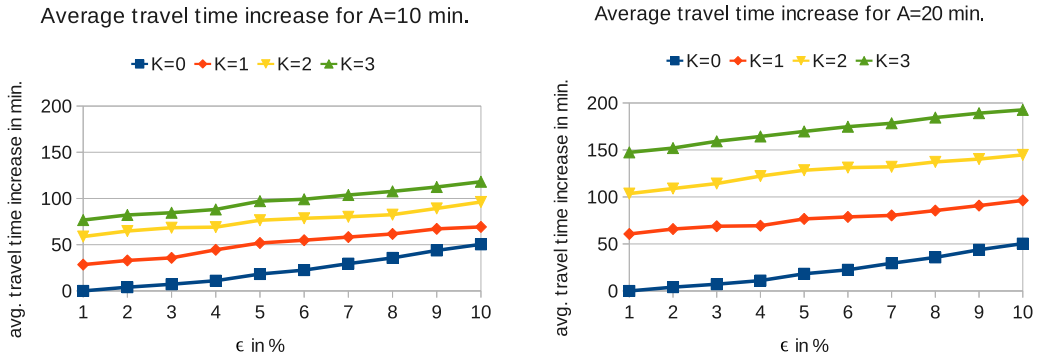

Figure 4 The absolute increase of travel time for quickest robust paths over optimal paths in the nominal scenario for different parameter sets of the delay scenarios.

strictly robust transfer arcs, we use our conservative over-approximation Algorithm 1 to compute the set $\mathcal{A}^{acc}$. The CPU time to compute $\mathcal{A}^{acc}$ is about 13 minutes per parameter set.

Experiment 2 — price of strict robustness. With this experiment we want to study quantitatively by how much the planned travel time increases when we compare strictly robust paths with nominal optimal paths. To this end, we have built 1000 random requests (the same set for each parameter setting; in our evaluations we always average over these requests). As a basis for our comparison, we determine for each request the earliest arrival time with respect to the planned schedule (nominal scenario). Among all paths with earliest arrival time we determine the minimum number of transfers. To solve these requests, we use a (standard) multi-criteria, time-dependent shortest path algorithm. Our implementation reuses the approach described in [3]. For the strictly robust requests the code has been extended to handle "forbidden transfers". More precisely, it is now possible to specify a list of forbidden transfers between pairs of trains, as computed in Experiment 1 by delay accumulation.

Figure 3 shows the average relative increase in travel time induced by strictly robust paths in comparison with optimal paths in the nominal scenario. The average travel time for the nominal paths is 456 minutes. This implies that the absolute average increase of the travel time in minutes becomes quite large — even for moderate parameter sets,

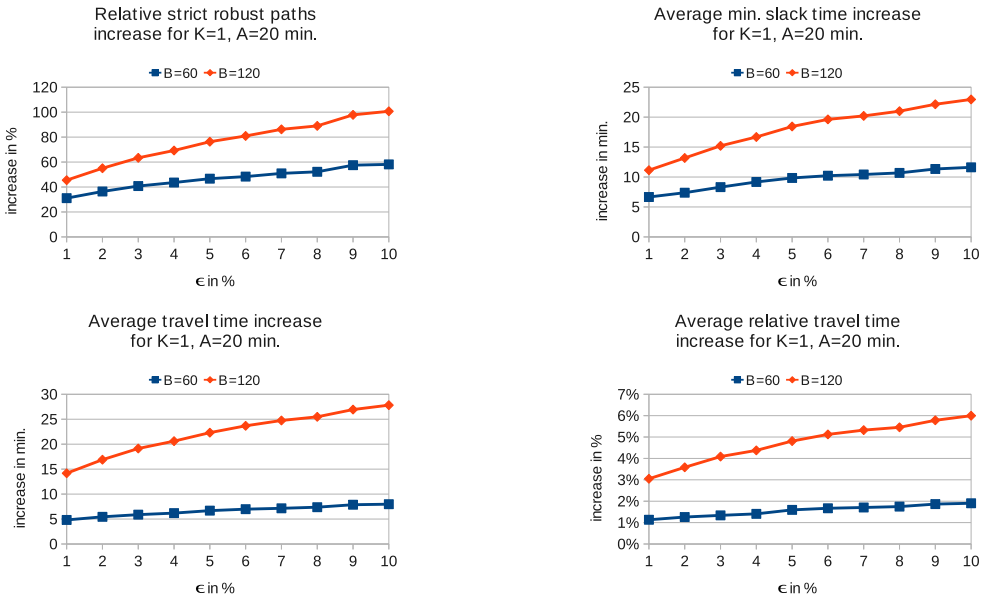

Figure 5 Light robustness: The increase of strict robust paths (in %, upper left), the increase in minimum slack times on the chosen light robust path in comparison with the nominal scenario (upper right), the average increase of travel time in minutes (lower left) and the corresponding percentage increase (lower right) for different parameter sets of delay scenarios.

see Figure 4. As expected, Figure 3 clearly shows that the price of robustness increases monotonously for increasing levels of guaranteed reliability, it grows roughly linearly with respect to parameter ε.

Experiment 3 — price of light robustness. Reusing the same set of random requests from Experiment 2, we analyzed the price of light robustness. The maximum increase of travel time over the nominal fastest one was bounded from above by the parameter B (in minutes), with $B \in \{60, 120\}$. The added value of a light robust solution in comparison with an optimal solution in the nominal scenario can be measured in two ways:

1. How often is the solution of the light robust optimization problem even a strictly robust one?
2. What is the effect on the minimum slack time for changing activities? This number tells us for each passenger the minimum buffer time available for his transfers.

Figure 5 shows the percentage increase of the number of cases where the light robust solution turns out to use only transfer arcs that have been recognized as strictly robust. The price to achieve this is a relatively moderate average increase of travel time — much more acceptable than for strict robustness (see lower part of the figure). We also evaluated by how much the minimum slack time for changing activities increases (upper right part of Figure 5) for light robust paths in comparison with the nominal case. This measure also clearly shows the added reliability achievable by light robustness.

7 Conclusion and future work

Two concepts for calculating robust passenger paths in public transportation networks are proposed: One that searches for routes that will *never fail* for a given set of delay scenarios, and one that finds the most reliable route within a given extra time. Both problems can be

solved efficiently when the set of strictly robust changing activities $\mathcal{A}^{SR}$ is known. However, determining this set is strongly NP-complete. We propose a dynamic programming algorithm to find an approximation of this set. In an experimental study, we quantitatively evaluated both robustness concepts using the approximate set of robust transfers. The trade-off between the wish to have more robust paths and the resulting travel time is shown for different levels of protection against delays.

Further research includes to improve our algorithms and to apply other robustness concepts, such as *recovery robustness* [5, 10] to the problem of finding robust passenger paths. Here, a solution does not need to be feasible for all scenarios, but whatever is going to happen, we want to have a *recovery algorithm* at hand which is able to repair the solution if the scenario becomes known.

References

1 A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, Princeton and Oxford, 2009.

2 A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

3 A. Berger, M. Grimmer, and M. Müller-Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest paths searches in time-dependent networks. In P. Festa, editor, *Proceedings of SEA 2010*, volume 6049 of *LNCS*, pages 35–46. Springer, Heidelberg, 2010.

4 D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

5 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 28–60. Springer, Heidelberg, 2009.

6 Y. Disser, M. Müller-Hannemann, and M. Schnee. Multi-criteria shortest paths in time-dependent train networks. In C. C. McGeoch, editor, *WEA 2008. 7th International Workshop on Experimental Algorithms, Provincetown, MA, USA*, volume 5038 of *LNCS*, pages 347–361. Springer, Heidelberg, 2008.

7 M. Fischetti and M. Monaci. Light robustness. In R. K. Ahuja, R.H. Möhring, and C.D. Zaroliagis, editors, *Robust and online large-scale optimization*, volume 5868 of *LNCS*, pages 61–84. Springer, Heidelberg, 2009.

8 M. Goerigk, M. Knoth, M. Müller-Hannemann, M. Schmidt, and A. Schöbel. The price of robustness in timetable information. Technical report, University Halle-Wittenberg, Institute of Computer Science, 2011.

9 P. Kouvelis and G. Yu. *Robust Discrete Optimization and its applications*. Kluwer, 1997.

10 C. Liebchen, M. Lübbecke, R. H. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In R. K. Ahuja, R.H. Möhring, and C.D. Zaroliagis, editors, *Robust and online large-scale optimization*, volume 5868 of *LNCS*, pages 1–27. Springer, Heidelberg, 2009.

11 M. Müller-Hannemann and M. Schnee. Efficient timetable information in the presence of delays. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer, Heidelberg, 2009.

12 M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4395 of *LNCS*, pages 67–89. Springer, Heidelberg, 2007.

13 M. Schnee. *Fully realistic multi-criteria timetable information systems*. PhD thesis, Fachbereich Informatik, Technische Universität Darmstadt, 2009. Published in 2010 by Südwestdeutscher Verlag für Hochschulschriften.

14 A.L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21:1154–1157, 1973.