Robust Routing in Urban Public Transportation: How to Find Reliable Journeys Based on Past Observations *

Kateřina Böhmová, Matúš Mihalák, Tobias Pröger, Rastislav Šrámek, and Peter Widmayer

Institute of Theoretical Computer Science, ETH Zurich, Switzerland
{kboehmov,mmihalak,tproeger,rsramek,widmayer}@inf.ethz.ch

Abstract

We study the problem of robust routing in urban public transportation networks. In order to propose solutions that are robust for typical delays, we assume that we have past observations of real traffic situations available. In particular, we assume that we have "daily records" containing the observed travel times in the whole network for a few past days. We introduce a new concept to express a solution that is feasible in any record of a given public transportation network. We adapt the method of Buhmann et al. [4] for optimization under uncertainty, and develop algorithms that allow its application for finding a robust journey from a given source to a given destination. The performance of the algorithms and the quality of the predicted journey are evaluated in a preliminary experimental study. We furthermore introduce a measure of reliability of a given journey, and develop algorithms for its computation. The robust routing concepts presented in this work are suited specially for public transportation networks of large cities that lack clear hierarchical structure and contain services that run with high frequencies.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory (Graph algorithms, Network problems), I.2.6 Learning

Keywords and phrases Algorithms, Optimization, Robustness, Route planning, Public transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2013.27

1 Introduction

We study the problem of routing in urban public transportation networks, such as tram and bus networks in large cities, focusing on the omnipresent uncertain situations when (typical) delays occur. In particular, we search for robust routes that allow reliable yet quick passenger transportation. We think of a "dense" tram network in a large city containing many tram lines, where each tram line is a sequence of stops that is served repeatedly during the day, and where there are several options to get from one location to another. Such a network usually does not contain clear hierarchical structure (as opposed to train networks), and each line is served with high frequency. Given two tram stops a and b together with a latest arrival time t_A, our goal is to provide a simple yet robust description of how to travel in the

* This work has been partially supported by the Swiss National Science Foundation (SNF) under the grant number 200021 138117/1, and by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS). Kateřina Böhmová is a recipient of the Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship.

given network from a to b in order to arrive on time t_A even in the presence of typical delays. We base our robustness concepts on past traffic data in a form of recorded timetables – the actually observed travel times of all lines in the course of several past days. If no delays occur, such a recorded timetable corresponds to the scheduled timetable for that day.

The standard approach to describe a travel plan from a to b in a given tram network is to specify, according to a scheduled timetable, the concrete sequence of vehicles together with transfer stops and departure/arrival times for each transfer stop. Such a travel plan may look like this: Take the tram 6 at 12:33 from stop a and leave it at 12:47 at transfer stop s; then take the tram 10 at 12:51 from s and leave it at 12:58 at b. However, such a travel plan may become infeasible on a concrete day due to delays: Imagine a situation where the tram 6 left a at 12:33, but arrived to s only at 12:53, and the tram 10 leaving s at 12:51 was on time. Then, the described travel plan would bring the passenger to stop s but it does not specify how to proceed further in order to arrive to b.

We observe that the standard solution concepts (such as paths in a time-expanded graph) are not suitable for our setting. We introduce a new concept to express a solution, which we call a *journey*, that is feasible in any recorded timetable of a given transportation network assuming the timetable to be periodic. A journey specifies an initial time t_D and then only a sequence of transportation lines $\langle l_1(\text{tram}), l_2(\text{bus}), \ldots, l_k(\text{tram})\rangle$ together with transfer stops $\langle s_1, \ldots, s_{k-1}\rangle$. This travel plan suggests to start waiting at a at time t_D, take the first tram of line l_1 that comes and travel to stop s_1, then change to the first coming bus of line l_2, etc. Since we assume that the frequency of vehicles serving each line is high, such a travel plan is not only feasible in our setting but also reasonable, and provides the passenger with all the necessary information. We provide algorithms to efficiently compute these journeys.

Equipped with the introduced solution concept of a journey, we can easily adapt the method of Buhmann et al. [4] for optimization under uncertainty, and apply it to identify robust travel plans. A key ingredient of the method is the ability to count the number of (possibly exponentially many) "good" solutions. Our solution concept allows us to develop efficient algorithms to compute the number of all journeys from a to b that depart after the time t_D and arrive before the time t_A.

Finally, we suggest an alternative simple measure for reliability of a given journey, expressed simply as the fraction of recorded timetables where the journey was successful and allowed to arrive at the destination on time. We provide efficient algorithms for computation of this measure.

2 Related Work

The problem of finding a fastest journey (according to the planned timetable) using public transportation has been extensively studied in the literature. Common approaches model the transportation network as a graph and compute a shortest path in this graph (see [12] for a survey). Various improvements have been developed, and experimental studies suggest that these can also be used in practice (see, e.g., [2, 5, 14]). Recent approaches avoid the construction of a graph and process the timetable directly [6, 7]. For example, Delling et al. [6] describe an approach which is centered around transportation lines (such as train or bus lines) and which can be used to find all pareto-optimal journeys when the arrival time and the number of stops are considered as criteria. Bast et al. [1] observe that for two given stops, we can find and encode each sequence of intermediate transfer stations (i.e., stations where we change from one line to another) that can lead to an optimal route. The set of these sequences of transfers is called *transfer pattern*. These patterns can be precomputed,

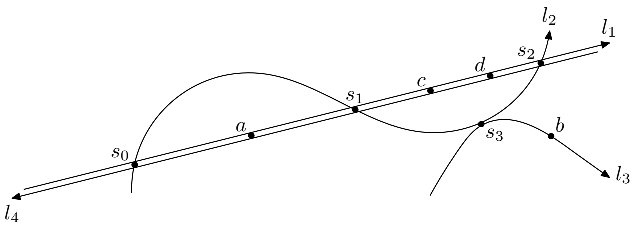

Figure 1 The line l_1 is a sequence of stops $\langle \dots, s_0, a, s_1, c, d, s_2, \dots \rangle$. The line $l_4 = \langle \dots, s_2, d, c, s_1, b, s_0, \dots \rangle$ that goes in the opposite direction to l_1 is considered to be a different line. In this example, both $a \triangleleft l_1$ and $a \triangleleft l_4$ hold, but $a \triangleleft l_2$ does not. Similarly, $s_1 \triangleleft s_2 \triangleleft l_1$ holds, but $s_1 \triangleleft s_2 \triangleleft l_4$ does not. The set $l_1 \cap l_2$ of all stops common to l_1 and l_2 is $\{s_0, s_1, s_2\}$. Moreover, when travelling from a to b using a route $\langle l_1, l_2, l_3 \rangle$, this network is an example where not every stop in $l_1 \cap l_2$ is suitable for changing from l_1 to l_2: We cannot choose s_0 as transfer stop since it is served before a. If s_2 was chosen, then l_3 can never be reached without travelling back. Thus, the only valid stop to change the line is s_1.

leading to very fast query times. These approaches are similar to our approach in the sense that they try to explicitly exploit the problem structure (e.g., by considering lines) instead of implicitly modelling all properties into a graph.

For computing *robust* journeys in public transportation, stochastic networks have been studied [3, 9, 13], where the delays between successive edges are random variables. Dibbelt et al. [7] study the case when stochastic delays on the vehicles are given. In a situation when timetables are fixed, Disser et al. [8] used a generalization of Dijkstra's algorithm to compute pareto-optimal multi-criteria journeys. They define the reliability of a journey as a function depending on the minimal time to change between two subsequent trains, and use it as an additional criterion. Müller-Hannemann and Schnee [11] introduced the concept of a *dependency graph* for a prediction of secondary delays caused by some current primary delays, which are given as input. They also show how to compute a journey that is optimal with respect to the predicted delays. Goerigk et al. [10] consider a given set of delay *scenarios* for every event, and adapt *strict robustness* to it, i.e. they aim to compute a journey that arrives on time for every scenario. Furthermore, the concept of *light robustness* is introduced, which aims to compute a journey that maximizes the number of scenarios in which the travel time of this journey lies at most a fixed time above the optimum. Strict robustness requires a feasible solution for every realization of delays for every event. This is quite conservative, as in reality not every combination of event delays appears. Our approach tries to avoid this by learning from the typical delay scenarios as recorded for each individual day.

3 Modeling issues

3.1 Model

Stops and lines Let $\mathcal{S}$ be a set of stops, and $\mathcal{L} \subset \bigcup_{i=2}^{|\mathcal{S}|} \mathcal{S}^i$ be a set of lines (e.g., bus lines, tram lines or lines of other means of transportation). The following basic definitions are illustrated in Figure 1. Every line $l \in \mathcal{L}$ is a sequence of $S(l)$ stops $\langle s_1^{(l)}, \dots, s_{S(l)}^{(l)} \rangle$, where, for every $i \in \{1, \dots, S(l) - 1\}$, the stop $s_i^{(l)}$ is served directly before $s_{i+1}^{(l)}$ by the line l. We explicitly distinguish two lines that serve the same stops but have opposite directions (these may be operated under the same identifier in reality). For a stop $s \in \mathcal{S}$ and a line $l \in \mathcal{L}$, we write $s \triangleleft l$ if s is a stop on the line l, i.e. if there exists an index $i \in \{1, \dots, S(l)\}$ such that $s = s_i^{(l)}$. Furthermore, for two stops $s_1, s_2 \in \mathcal{S}$ and a line $l \in \mathcal{L}$ we write $s_1 \triangleleft s_2 \triangleleft l$ if both s_1 and s_2 are stops on l and s_1 is served before s_2, i.e. if there exist indices $i, j \in \mathbb{N}$, $1 \leq i \leq S(l) - 1$, $i + 1 \leq j \leq S(l)$ such that $s_1 = s_i^{(l)}$ and $s_2 = s_j^{(l)}$. For two lines $l_1, l_2 \in \mathcal{L}$, we define $l_1 \cap l_2$ to be the set of all stops $s \in \mathcal{S}$ that are served both by l_1 and l_2.

Trips and timetables While the only information associated with a line itself are its consecutive stops, it usually is operated multiple times per day. Each of these concrete realizations that departs at a given time of the day is called a *trip*. With every trip τ we associate a line $L(\tau) \in \mathcal{L}$. By $L^{-1}(l)$ we denote the set of *all* trips associated with a line $l \in \mathcal{L}$. For a trip τ and a stop $s \in \mathcal{S}$, let $A(\tau, s)$ be the arrival time of τ at stop s, if $s \triangleleft L(\tau)$. Analogously, let $D(\tau, s)$ be the departure time of τ at s. In the following, we assume time to be modelled by integers. For a given trip τ, we require $A(\tau, s) \leq D(\tau, s)$ for every stop $s \in L(\tau)$. Furthermore we require $D(\tau, s_1) \leq A(\tau, s_2)$ for every two stops $s_1, s_2 \in \mathcal{S}$ with $s_1 \triangleleft s_2 \triangleleft L(\tau)$. A set of trips is called a *timetable*. We distinguish between

1. the *planned* timetable T. We assume it to be periodic, i.e., every line realized by some trip τ will be realized by a later trip τ' again (probably not on the same day).
2. *recorded* timetables T_i that describe how various lines were operated during a given time period (i.e., on a concrete day or during a concrete week). These recorded timetables are concrete executions of the planned timetable.

In the following, *timetable* refers both to the planned as well as to a recorded timetable.

Goal In the following, let $a, b \in \mathcal{S}$ be two stops, $m \in \mathbb{N}_0$ be the maximal allowed number of line changes, and $t_A \in \mathbb{N}$ be the latest arrival time. A *journey* consists of a departure time t_D, a sequence of lines $\langle l_1, \dots, l_k \rangle$, $k \leq m + 1$ and a sequence of transfer stops $\langle s_{\mathrm{CH}}^{(1)}, \dots, s_{\mathrm{CH}}^{(k-1)} \rangle$. The intuitive interpretation of such a journey is to start at stop a at time t_D, take the first line l_1 (more precisely, the first available trip of the line l_1), and for every $i \in \{1, \dots, k-1\}$, leave l_i at stop $s_{\mathrm{CH}}^{(i)}$ and take the next arriving line l_{i+1} immediately. Our goal is to compute a recommendation to the user in form of one or more (robust) journeys from a to b that will likely arrive on time (i.e., before time t_A) on a day for which the concrete travel times are not known yet. We formalize the notion of robustness later. We note that for the convenience of the user, one should handle two different lines l_1 and l_2 operating between two stops s_1 and s_2 as one (virtual) line, and provide recommendations of the form "in s_1, take the first line l_1 or l_2 to s_2, etc.". For the sake of simplicity we do not pursue this generalization further, but will consider this in the future.

Routes Let $k \in \{1, \dots, m + 1\}$ be an integer. A sequence of lines $r = \langle l_1, \dots, l_k \rangle \in \mathcal{L}^k$ is called a *feasible route from* a *to* b if there exist $k + 1$ stops $s_0 := a, s_1, \dots, s_{k-1}, s_k := b$ such that $s_{i-1} \triangleleft s_i \triangleleft l_i$ for every $i \in \{1, \dots, k\}$, i.e., if both s_{i-1} and s_i are stops on line l_i, and s_{i-1} is served before s_i on line l_i. Notice that on a feasible route $r \in \mathcal{L}^k$ we need to change the line at $k - 1$ transfer stops. Let

$$\mathcal{R}_{ab}^m = \{r \in \mathcal{L} \cup \mathcal{L}^2 \cup \dots \cup \mathcal{L}^{m+1} \mid r \text{ is a feasible route from } a \text{ to } b\} \tag{1}$$

be the set of all feasible routes from a to b using at most m transfer stops. If a, b and m are clear from the context, for simplicity we just write $\mathcal{R}$ instead of $\mathcal{R}_{ab}^m$. Notice that by definition, a line l may occur multiple times in a route. This is reasonable because there might be two transfer stops s, s' on l and one or more intermediate lines that travel faster from s to s' than l does. Additionally, notice that a route does not contain *any* time information.

3.2 Computation of Feasible Routes

Input data In this section we describe an algorithm that, given a set of stops $\mathcal{S}$ and a set of lines $\mathcal{L}$, finds the set $\mathcal{R}$ of all feasible routes that allow to travel from a given initial stop a to a given destination stop b using at most m transfer stops (also called transfers). Notice that to compute $\mathcal{R}$ we only need the network structure, no particular timetable is necessary.

Preprocessing the input We preprocess the input data and construct data structures to allow efficient queries of the following types:

1. $Q(l, s)$: *Compute the position of s on l.* Given a line $l = \langle s_1, \ldots, s_{|S(l)|}\rangle$, $Q(l, s)$ returns j if s is the j-th stop on l, i.e., if $s = s_j$, or 0 if s is not served by l.
2. $Q(l, s_i, s_j)$: *Determine whether s_i is served before s_j on l.* Given a line l and two stops s_i, s_j, $Q(l, s_i, s_j)$ returns `TRUE` iff $s_i, s_j \lhd l$ and $s_i \lhd s_j \lhd l$.
3. $Q(l_i, l_j)$: *Determine $l_i \cap l_j$ (i.e., the stops shared by l_i and l_j) in a compact, ordered format.* Given two lines l_i, and l_j, $Q(l_i, l_j)$ returns the set $l_i \cap l_j$ of stops shared by these lines. We encode $l_i \cap l_j$ into an ordered set I_{ij} of pairs of stops with respect to l_i in such a way that $(s_q, s_r) \in I_{ij}$ indicates that l_i and l_j share the stops s_q, s_r, and all the stops in between on the line l_i (independent of their order on l_j). Thus, $Q(l_i, l_j)$ outputs the described sorted set I_{ij} of pairs of stops. The motivation to compress $l_i \cap l_j$ into I_{ij} is that, in practice, there may be many stops shared by l_i and l_j, but only a small number of contiguous intervals of such stops. Notice that $Q(l_i, l_j)$ doesn't need to be equal to $Q(l_j, l_i)$, nor the sequence in reverse order; an example is given in Figure 2.

Notice that these queries can be answered in expected constant time if implemented using suitable arrays or hashing tables.

Graph of line incidences The function $Q(l_i, l_j)$ induces the following directed graph G. The set V of vertices of G corresponds to the set of lines $\mathcal{L}$. There is an edge from a vertex (line) l_i to a vertex l_j if and only if $Q(l_i, l_j) \neq \emptyset$. Then, $Q(l_i, l_j)$ is represented as a tag of the edge (l_i, l_j). We construct and represent the graph G as adjacency lists.

Preliminary observations Given two stops a and b, and a number m, we want to find all routes $\mathcal{R}$ that allow to travel from a to b using at most m transfers in the given public transportation network described by a set of stops $\mathcal{S}$ and a set of lines $\mathcal{L}$. Notice that each such route $r = \langle l_1, \ldots, l_k\rangle \in \mathcal{L}^k$ with $0 < k \leq m + 1$ has the following properties.

1. Both $Q(l_1, a)$ and $Q(l_k, b)$ are nonzero (i.e., $a \lhd l_1$, and $b \lhd l_k$).
2. The vertices $l_1, \ldots, l_k$ form a path in G (i.e., $l_i \cap l_{i+1} \neq \emptyset$ for every $i = 1, \ldots, k - 1$).
3. There exists a sequence of stops $a = s_0, s_1, \ldots, s_{k-1}, s_k = b$ such that $Q(l_i, s_{i-1}, s_i)$ is `TRUE` (i.e., $s_{i-1} \lhd s_i \lhd l_i$) for every $i = 1, \ldots, k$.

These observations lead to the following algorithm to find the set of routes $\mathcal{R}$.

All routes algorithm For the stop a, determine the set $\mathcal{L}_a$ of all lines passing through a. Then explore the graph G from the set $\mathcal{L}_a$ of vertices in the following fashion. For each vertex $l_1 \in \mathcal{L}_a$, perform a depth-first search in G up to the depth m, but do not stop when finding a vertex that has already been found earlier. In each step, try to extend a partial path $\langle l_1, \ldots, l_j\rangle$ to a neighbor l'_j of l_j in G. Keep track of the *current transfer stop* s_q. This is a stop on the currently considered line l_j such that s_q is the stop with the smallest position on l_j at which it is possible to transfer from l_{j-1} to l_j, considering the partial path from l_1 to l_{j-1}. In other words, s_q is the stop on the considered route where the line l_j can be boarded. Each step of the algorithm is characterized by a *search state*: a partial path $P = \langle l_1, \ldots, l_j\rangle$, and a current transfer stop s_q that allowed the transfer to line l_j. The initial search state consists of the partial path $P = \langle l_1\rangle$ and the current transfer stop a. More specifically, to

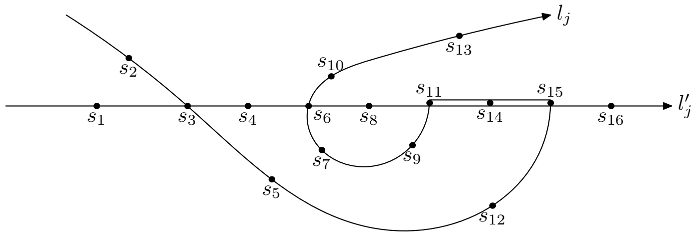

Figure 2 Lines l_j and l'_j have common stops s_3, s_6, s_{11}, s_{14}, and s_{15}. The ordered set $I_{jj'} = Q(l_j, l'_j)$ consists of the pairs $\{(s_3, s_3), (s_{15}, s_{11}), (s_6, s_6)\}$. Thus, the last stop in the last interval of $I_{jj'}$ is the stop s_6. On the other hand, the ordered set $I_{j'j} = Q(l'_j, l_j)$ consists of the pairs $\{(s_3, s_3), (s_6, s_6), (s_{11}, s_{15})\}$. Now, imagine that the current transfer stop s_q for a partial path $P = \langle l_1, \ldots, l_j\rangle$ is s_2, then the stop s_3 is the current transfer stop s'_q for a partial path $P' = \langle l_1, \ldots, l_j, l'_j\rangle$. However, observe that if s_q is s_{12}, then s'_q needs to be s_6.

process a search state with the partial path $P = \langle l_1, \ldots, l_j\rangle$, and the current transfer stop s_q, perform the following tasks:

1. Check whether the line corresponding to the vertex l_j contains the stop b and whether s_q is before b on l_j. If this is the case (i.e., the query $Q(l_j, s_q, b)$ returns `TRUE`), then the partial path P corresponds to a feasible route and is output as one of the solutions in $\mathcal{R}$.
2. If the partial path P contains at most $m-1$ edges (thus the corresponding route has at most $m-1$ transfers, and can be extended), then for each neighbor l'_j of l_j check whether extending P by l'_j is possible (and if so, update the current transfer stop) as follows. Let $I_{jj'} = Q(l_j, l'_j)$ be the set of pairs of stops sorted as described in the previous section. Recall that each pair $(s_u, s_v) \in I_{jj'}$ encodes an interval of one or several consecutive stops on l_j that are also stops on the line l'_j. Let s_z be the last stop in the last interval of $I_{jj'}$. Similarly, let $I_{j'j} = Q(l'_j, l_j)$. If $Q(l_j, s_q, s_z)$ is `TRUE`, then $s_q \lhd s_z \lhd l_j$, and the path P can be extended to l'_j.
   a. We determine the current transfer stop s'_q for l'_j by considering the pairs/intervals of $I_{j'j}$ in ascending order and deciding whether the position of s_q on the line l_j is before one of the endpoints of the currently considered interval. We refer to Figure 2 for a nontrivial case of computing of the current transfer stop.
   b. Perform the depth search with the search state consisting of the partial path $P' = \langle l_1, \ldots, l_j, l'_j\rangle$ and the current transfer stop s'_q.

   Otherwise, if $Q(l_j, s_q, s_z)$ is `FALSE`, it is not possible to extend P to l'_j.

The theoretical running time of the algorithm is $\mathcal{O}(\Delta^m)$, where Δ is the maximum degree of G. However, we believe that in practice the actual running time will rather linearly correspond to the size of the output $\mathcal{O}(m|\mathcal{R}|)$. On real-world data, the algorithm performs reasonably fast (see section 6 for details).

3.3 Computing the earliest arriving journey

Recursive computation As previously stated, let $a \in \mathcal{S}$ be the initial stop, $b \in \mathcal{S}$ be the destination stop, $\varepsilon(s, l, l')$ be the minimum time to change from line l to line l' at station s, and $t_A \in \mathbb{N}$ be the latest arrival time. In the previous section we showed how the set $\mathcal{R}$ of all feasible routes from a to b can be computed. However, instead of presenting just a route $r \in \mathcal{R}$ to the user, our final goal is to compute a departure time t_0 and a *journey* that arrives at b before time t_A. For the following considerations, we assume the underlying timetable (either the planned or a recorded timetable) to be fixed. Given $a, b \in \mathcal{S}$, an initial departure time $t_0 \in \mathbb{N}$, and a route $r = \langle l_1, \ldots, l_k \rangle \in \mathcal{R}$, a journey along r that arrives as early as possible can be computed as follows. We start at a at time t_0 and take the first line l_1 that arrives. Then we compute an appropriate transfer stop $s \in l_1 \cap l_2$ (that is served both by l_1 as well as by l_2) and the arrival time t_1 at s, leave l_1 there and compute recursively the earliest arrival time when departing from s at time at least $t_1 + \varepsilon(s, l_1, l_2)$, following the route $\langle l_2, \ldots, l_k \rangle$. Notice that the selection of an appropriate transfer stop s is the only non-trivial part due to mainly two reasons:

1. The lines l_1 and l_2 may operate with different speeds (e.g., because l_1 is a fast tram while l_2 is a slow bus), or l_1 and l_2 separate at a stop s_1 and join later again at a stop s_2 but the overall travel times of l_1 and l_2 differ between s_1 and s_2. Depending on the situation, it may be better to leave l_1 as soon or as late as possible, or anywhere inbetween.
2. The lines l_1 and l_2 may separate at a stop s_1 and join later again at a stop s_2. If all transfer stops in $l_2 \cap l_3$ are served by l_2 before s_2, then leaving l_1 at s_2 is not an option since l_3 is not reachable anymore. See Figure 1 for a visualization.

The idea now is to find the earliest trip of line l_1 that departs from a at time t_0 or later, iterate over all stops $s \in l_1 \cap l_2$, and compute recursively the earliest arrival time when continuing the journey from s having a changing time of at least $\varepsilon(s, l_1, l_2)$. Finally, we return the smallest arrival time that was found in one of the recursive calls.

Issues and improvement of the recursive algorithm An issue with this naïve implementation is the running time, which might be exponential in k in the worst-case (if $|l_i \cap l_{i+1}| > 1$ for $\Omega(k)$ many $i \in \{1, \ldots, k-1\}$). Let τ and τ' be two trips with $L(\tau) = L(\tau')$. If τ leaves before τ' at some stop s, we assume that it will never arrive later than τ' at any subsequent stop s', $s \lhd s' \lhd L(\tau)$, i.e. consecutive trips of the same line do not overtake. For a line $l \in \mathcal{L}$ and a set of trips $T_l \subseteq L^{-1}(l)$, it follows that taking the earliest trip in T_l never results in a later arrival at b than taking any other trip from T_l. Furthermore, a trip $\tau \in T_l$ is operated earlier than a trip $\tau' \in T_l$ iff $A(\tau, s) < A(\tau', s)$ for *any* stop $s \lhd l$.

Thus, we can iterate over some appropriate stops in $l_1 \cap l_2$ to find the earliest reachable trip associated with l_2. We just need to ignore those stops where changing to l_3 is no longer possible (see Figure 1 for an example).

Computing appropriate transfer stops The problem to find these appropriate stops can be solved by first sorting $l_1 \cap l_2 = \{s_1, \ldots, s_n\}$ such that $s_j \lhd s_{j+1} \lhd l_1$ for every $j \in \{1, \ldots, n-1\}$. Obviously, all stops that appear before a on line l_1 cannot be used for changing to l_2. This problem can easily be solved by considering only those stops s_j where $a \lhd s_j \lhd l_1$. Unfortunately, the last $m \geq 0$ stops $s_{n-m+1}, \ldots, s_n$ might also not be suitable for changing to l_2 because they may prevent us later to change to some line l_j (e.g., if *all* stops of $l_2 \cap l_3$ are served before $s_{n-g+1}, \ldots, s_n$ on l_2, then changing to l_3 is no longer possible). We solve this problem by precomputing (the index of) the last stop s_j where all later lines are still reachable. This can be done backwards: we start at b, order the elements of $l_k \cap l_{k-1}$ as they appear on line l_k, and find the last stop that is served before b on l_k. We recursively continue with $l_1, \ldots, l_{k-1}$ and use the stop previously computed as the stop that still needs to be reachable.

Iterative algorithm The improved algorithm first iterates over $i \in \{1, \ldots, k-1\}$, and uses the aforementioned algorithm to precompute the index last$[i]$ of the last stop where changing from l_i to l_{i+1} is still possible (with respect to the route $\langle l_1, \ldots, l_k \rangle$). After that, for every $i \in \{1, \ldots, k-1\}$, we iterate over the appropriate transfer stops $s \in l_i \cap l_{i+1}$ where changing to l_{i+1} is possible, and find among those the stop $s_{\mathrm{CH}}^{(i)}$ where the earliest trip τ_{i+1} associated with line l_{i+1} departs. Finally we obtain a sequence of trips $\tau_1, \ldots, \tau_k$ along with transfer stops $s_{\mathrm{CH}}^{(0)} := a, s_{\mathrm{CH}}^{(1)}, \ldots, s_{\mathrm{CH}}^{(k)}$ to change lines. Since we gradually compute the earliest trips τ_i for each of the lines l_i, the earliest time to arrive at b is simply $A(\tau_k, b)$.

EarliestArrival$(a, b, t_0, \langle l_1, \ldots, l_k \rangle)$

last$[k] \leftarrow b$
for $i \leftarrow k, \ldots, 2$ **do**
    Order the elements of $l_i \cap l_{i-1} = \{s_1, \ldots, s_n\}$ s.t. $s_j \lhd s_{j+1} \lhd l_{i-1}$ $\forall j \in \{1, \ldots, n-1\}$.
    last$[i-1] \leftarrow \max\{j \in \{1, \ldots, n\} \mid s_j \lhd \text{last}[i] \lhd l_i\}$
$\tau_1 \leftarrow \arg\min_{\tau \in L^{-1}(l_1)}\{D(\tau, a) \mid D(\tau, a) \geq t_0\}$; $s_{\text{CH}}^{(0)} \leftarrow a$
for $i \leftarrow 1, \ldots, k-1$ **do**
    Order the elements of $l_i \cap l_{i+1} = \{s_1, \ldots, s_n\}$ s.t. $s_j \lhd s_{j+1} \lhd l_i$ $\forall j \in \{1, \ldots, n-1\}$.
    $\tau_{i+1} \leftarrow$ **null**; $s_{\text{CH}}^{(i)} \leftarrow$ **null**; $A_{s_n}^{(i+1)} \leftarrow \infty$
    for $j \leftarrow 1, \ldots,$ last$[i]$ **do**
        if $s_{\text{CH}}^{(i-1)} \lhd s_j \lhd l_i$ **and** $s_j \lhd \text{last}[i+1] \lhd l_{i+1}$ **then**
            $\tau' \leftarrow \arg\min_{\tau \in L^{-1}(l_{i+1})}\{D(\tau, s_j) \mid D(\tau, s_j) \geq A(\tau_i, s_j) + \varepsilon(s_j, l_i, l_{i+1})\}$
            if $A(\tau', s_n) < A_{s_n}^{(i+1)}$ **then** $\tau_{i+1} \leftarrow \tau'$; $s_{\text{CH}}^{(i)} \leftarrow s_j$; $A_{s_n}^{(i+1)} \leftarrow A(\tau', s_n)$
return $A(\tau_k, b)$

Let $n = \max\{|l_i \cap l_{i+1}|\}$. Given a line $l \in \mathcal{L}$, a station $s \in \mathcal{S}$ and a time $t_0 \in \mathbb{N}$, let f be the time to find the earliest trip τ with $L(\tau) = l$ und $D(\tau, s) \geq t_0$ (this time depends on the concrete implementation of the timetable). It is easy to see that the running time of the above algorithm is bounded by $\mathcal{O}(kn(\log n + f))$.

4 Maximizing the Unexpected Similarity

Computing the optimum journey for a fixed timetable Given two stops $a, b \in \mathcal{S}$ and a departure time $t_0 \in \mathbb{N}$, we can already compute the earliest arrival of a journey from a to b starting at time t_0. From now on, we aim to compute the latest departure time at a when the latest *arrival* time t_A at b is given. For this purpose we present an algorithm that sweeps backwards in time and uses the previous algorithm Earliest-Arrival. This sweepline algorithm will later be extended to count journeys (instead of computing a single one) and can be used for finding robust journeys, i.e. journeys that are likely to arrive on time.

The sweepline algorithm works as follows. We consider the trips departing at stop a before time t_A, sorted in reverse chronological order. Everytime we find a trip τ of any line departing at some time t_0, we check whether there exists a route $r = \langle L(\tau), l_2, \ldots, l_k \rangle \in \mathcal{R}$ that starts with the line $L(\tau)$. If yes, then we use the previous algorithm to compute the earliest arrival time at b when we depart at a at time t_0 and follow the route r. If the time computed is not later than t_A, we found the optimal solution and stop the algorithm. Otherwise we continue with the previous trip departing from a.

Finding robust journeys We will now describe how to compute robust journeys using the approach of Buhmann et al. [4]. We stress up front that this is "learning"-style algorithm and that it, in particular, does not specifically aims at optimizing some "robustness" criterion (such as the fraction of successes in the recorded timetables). Let $a, b \in \mathcal{S}$ be the departure and the target stop of the journey, t_A be the latest arrival time at b, and $\mathcal{T}$ be a set of recorded timetables for comparable time periods (e.g., daily recordings for the past Mondays). For a timetable $T \in \mathcal{T}$ and a value γ, the *approximation set* $A_\gamma(T)$ contains a route $r \in \mathcal{R}$ iff there exists a journey along the route r that starts at a at time $t_A - \gamma$ or later and arrives at b at time t_A or earlier (both times refer to timetable T). The major advantage of this definition over classical approximation definitions (such as multiplicative approximation) is

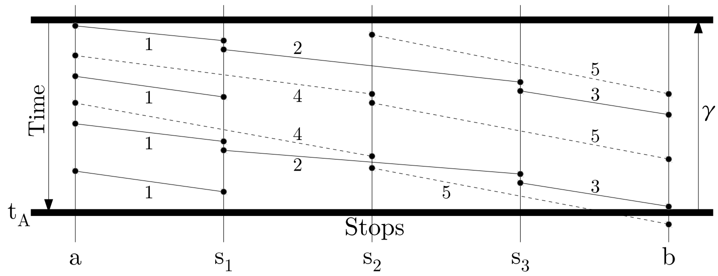

Figure 3 An example with five lines $\{1,\ldots,5\}$ and two routes $r_1 = \langle 1,2,3\rangle$ (solid) and $r_2 = \langle 4,5\rangle$ (dotted). The x-axis illustrates the stops $\{a, s_1, s_2, s_3, b\}$, whereas the y-axis the time. If a trip leaves a stop s_d at time t_d and arrives at a stop s_a at time t_a, it is indicated by a line segment from (s_d, t_d) to (s_a, t_a). We have $\mu^T_\gamma(r_1) = 3$ and $\mu^T_\gamma(r_2) = 1$.

that we can consider multiple recorded timetables at the same time, and that the parameter γ still has a direct interpretation as the time that we depart before t_A. Especially, if we consider approximation sets $A_\gamma(T_1),\ldots,A_\gamma(T_k)$ for $T_1,\ldots,T_k \in \mathcal{T}$, every set contains only routes that appear in the same time period and are therefore comparable among different approximation sets.

To identify *robust* routes when only two timetables $T_1, T_2 \in \mathcal{T}$ are given, we consider $A_\gamma(T_1) \cap A_\gamma(T_2)$: the only chance to find a route that is likely to be good in the future is a route that was good in the past for *both* recorded timetables. The parameter γ determines the size of the intersection: if γ is too small, the intersection will be empty. If γ is too large, the intersection contains many (and maybe all) routes from a to b, and not all of them will be a good choice. Assuming that we knew the optimal parameter γ_{OPT}, we could pick a route from $A_{\gamma_{\mathrm{OPT}}}(T_1) \cap A_{\gamma_{\mathrm{OPT}}}(T_2)$ at random. Buhmann et al. [4] suggest to set γ_{OPT} to the value γ that maximizes the so-called *similarity*

$$S_\gamma = \frac{|A_\gamma(T_1) \cap A_\gamma(T_2)|}{|A_\gamma(T_1)||A_\gamma(T_2)|}. \tag{2}$$

Notice that up to now we did not consider how often a route is realized by a journey in a recorded timetable. This is undesirable from a practical point of view: when we pick a route from $A_{\gamma_{\mathrm{OPT}}}(T_1) \cap A_{\gamma_{\mathrm{OPT}}}(T_2)$ at random, the probability to obtain a route should depend on how frequently it is realized. Therefore we change the definition of $A_\gamma(T)$ to a *multiset* of routes, and $A_\gamma(T)$ contains a route r as often as it is realized by a journey starting at time $t_A - \gamma$ or later, and arriving at time t_A or earlier. Figure 3 shows an example with five lines $\{1,\ldots,5\}$ and two routes $r_1 = \langle 1,2,3\rangle$ and $r_2 = \langle 4,5\rangle$. We have $\mu^T_\gamma(r_1) = 3$: taking the second 1 and the second 2 (from above) as well as taking the third 1 and the second 2 are counted as different journeys since the departure times at a differ. On the other hand, by our definition of journey we have to take the first occurence of a line that arrives, thus taking the first 1 and waiting for the second 2 is *not* counted.

Now the approximation set $A_\gamma(T)$ can be represented by a function $\mu^T_\gamma : \mathcal{R} \to \mathbb{N}_0$, where for a route $r \in \mathcal{R}$, $\mu^T_\gamma(r)$ is the number of journeys starting at time $t_A - \gamma$ or later, arriving at time t_A or earlier and following the route r. Thus, we have $|A_\gamma(T)| = \sum_{r\in\mathcal{R}} \mu^T_\gamma(r)$, and for two recorded timetables T_1, T_2, we need to compute

$$\gamma_{\mathrm{OPT}} = \arg\max_\gamma \frac{\sum_{r\in\mathcal{R}} \min(\mu^{T_1}_\gamma(r), \mu^{T_2}_\gamma(r))}{\left(\sum_{r\in\mathcal{R}} \mu^{T_1}_\gamma(r)\right) \cdot \left(\sum_{r\in\mathcal{R}} \mu^{T_2}_\gamma(r)\right)}. \tag{3}$$

After computing the value γ_{OPT}, we pick a route r from $A_{\gamma_{\text{OPT}}}(T_1) \cap A_{\gamma_{\text{OPT}}}(T_2)$ at random according to the probability distribution defined by

$$p_r := \frac{\min(\mu^{T_1}_{\gamma_{\text{OPT}}}(r), \mu^{T_2}_{\gamma_{\text{OPT}}}(r))}{\sum_{r\in\mathcal{R}} \min(\mu^{T_1}_{\gamma_{\text{OPT}}}(r), \mu^{T_2}_{\gamma_{\text{OPT}}}(r))}, \tag{4}$$

and search in the planned timetable for a journey from a to b that departs at time $t_A - \gamma_{\text{OPT}}$ or earlier, and that arrives at time t_A or earlier.

Computing the similarity For $i \in \{1, 2\}$, we represent the function $\mu^{T_i}_{\gamma}$ by an $|\mathcal{R}|$-dimensional vector μ_i such that $\mu_i[r] = \mu^{T_i}_{\gamma}(r)$ for every $r \in \mathcal{R}$. We can compute the value γ_{OPT} by a simple extension of the aforementioned sweepline algorithm. The modified algorithm again starts at time t_A, and considers all trips in T_1 and T_2 in reverse chronological order. The sweepline stops at every time when one or more trips in T_1 or in T_2 depart. Assume that the sweepline stops at time $t_A - \gamma$, and assume that it stopped at time $t_A - \gamma' > t_A - \gamma$ in the previous step. Of course, we have $\mu^{T_i}_{\gamma}(r) \geq \mu^{T_i}_{\gamma'}(r)$ for every $r \in \mathcal{R}$ and $i \in \{1, 2\}$. Let $\tau_1, \ldots, \tau_k$ be the trips that depart in T_1 or T_2 at time $t_A - \gamma$. The idea is to compute the values of μ_i (representing $\mu^{T_i}_{\gamma}$) from the values computed in the previous step (representing $\mu^{T_i}_{\gamma'}$). This can be done as follows: for every trip τ_j occuring in T_i and departing at time $t_A - \gamma$, we check whether there exists a route $r \in \mathcal{R}$ starting with $L(\tau_j)$. If yes, we distinguish two cases:

1. If $\mu_i[r] = 0$, then $\mu^{T_i}_{\gamma'}(r) = 0$, thus $r \notin A_{\gamma'}(T_i)$. If there exists a journey from a to b along r departing at time $t_A - \gamma$ or later, and arriving at time t_A or earlier, then $A_\gamma(T_i)$ contains r exactly once. Thus, if EARLIEST-ARRIVAL$(a, b, t_A - \gamma, r) \leq t_A$, we set $\mu_i[r] \leftarrow 1$.
2. If $\mu_i[r] > 0$, then $\mu^{T_i}_{\gamma'}(r) > 0$, thus $A_{\gamma'}(T_i)$ contains r at least once. Thus, there exists a journey from a to b along r departing at time $t_A - \gamma'$ or later, and arriving at time t_A or earlier. Since τ_i is the only possibility to depart at a between time $t_A - \gamma$ and $t_A - \gamma'$, τ_i is the first trip on a journey we never found before. Therefore it is sufficient to simply increase $\mu_i[r]$ by 1.

Up to now, we did not define when the algorithm terminates. In fact we stop if γ exceeds a value γ_{MAX}. Let $t_A - \gamma_i$ be the starting time of an optimal journey in T_i. Of course, γ_{MAX} has to be larger than $\max\{\gamma_1, \gamma_2\}$. In our experimental evaluation, we set γ_{MAX} to be one hour before t_A; good choices for γ_{MAX} will be investigated in further experiments.

5 Journey Reliability

Success rate as reliability Having several recorded timetables at our disposal, and a journey from a to b, a natural approach to assess its *reliability* with respect to the given latest arrival time t_A is to check how many times in the past the journey finished before t_A. Normalized by the total number of recorded timetables, we call this success rate the *coupled reliability*. This is the least information about robustness one would wish to obtain from online routing services when being presented, upon a query to the system, with a set of routes from a to b.

Few recorded timetables The generalizing expressiveness of coupled reliability is limited (and biased towards outliers in the samples) if the number of recorded timetables is small. If lines in the considered transportation network suffer from delays (mostly) independently, we can heuristically extract from each of the m given recorded timetables $T_1, \ldots, T_m$ an individual timetable $T(i, l)$ for every line l (storing just the travelled times of the specific line l in timetable T_i), and then evaluate the considered journey on every relevant combination of these individual *decoupled* timetables. This enlarges the number of evaluations of the journey and thus has a chance to better generalize/express the observed travel times as typical situation.

Decoupling the timetables We can formally describe this process as follows. We consider m recorded timetables $T_1, \dots, T_m$, and we consider a journey J from stop a to stop b, specified by a departure time t_D, by a sequence of lines $\langle l_1, \dots, l_k \rangle$, and by a sequence of transfer stops $\langle s_{\mathrm{CH}}^{(1)}, \dots, s_{\mathrm{CH}}^{(k-1)} \rangle$.

We say that journey J is *realizable in* $\langle T(i_1, l_1), T(i_2, l_2), \dots, T(i_k, l_k) \rangle$, $i_1, \dots, i_k \in \{1, \dots, m\}$, *with respect to a given latest arrival time* t_A, if for every line l_j there exists a trip t_j (of the line l_j) in $T(i_j, l_j)$ such that

1. The departure time of trip t_1 from stop a is after t_D,
2. the arrival time of trip t_k at stop b is before t_A, and
3. for every $j = 1, \dots, k-1$, the arrival time of trip t_j at stop $s_{\mathrm{CH}}^{(j)}$ is before the departure time of trip t_{j+1} at the same stop.

Decoupled reliability Clearly, there are m^k ways to create a k-tuple $\langle T(i_1, l_1), \dots, T(i_k, l_k) \rangle$. Let M denote the number of those k-tuples in which journey J is realizable with respect to a given t_A. We call the ratio $\frac{M}{m^k}$ the *decoupled reliability* of journey J with respect to the latest arrival time t_A.

Computational issues Computing the coupled reliability is very easy: For every timetable $T_i \in \{T_1, \dots, T_m\}$ we need to check whether the journey in question finished before time t_A or not. This can be done by a simple linear time algorithm that simply "simulates" the journey in the timetable T_i, and checks whether the arrival time of the journey lies before or after t_A. The computation of decoupled reliability is not so trivial anymore, as the straightforward approach would require to enumerate all m^k k-tuples $\langle T(i_1, l_1), \dots, T(i_k, l_k) \rangle$, and thus an exponential time. In the following section, we present an algorithm that avoids such an exponential enumeration.

Computing decoupled reliability We can reduce the enumeration of all k-tuples $\langle T(i_1, l_1), T(i_2, l_2), \dots, T(i_k, l_k) \rangle$ by observing that the linear order of the lines in journey J allows to use dynamic-programming. Let us denote for simplicity the boarding, transfer, and arrival stops of journey J as $s_0, s_1, \dots, s_k$, where $s_0 = a$, $s_k = b$, and $s_j = s_{\mathrm{CH}}^{(j)}$ for $j = 1, \dots, k-1$. For every stop s_{j-1}, $j = 1, \dots, k$, we store for every time event t of a departing trip τ of line l_j (in any of the timetables $T_1, \dots, T_m$) a "success rate" of the journey J: the fraction $SR[s_{j-1}, t]$ of all tuples $\langle T(i_j, l_j), \dots, T(i_k, l_k) \rangle$ in which the sub-journey of J from s_{j-1} to s_k starting at time t is realizable. For time t not being a departure event, we extend the definition and set $SR[s_{j-1}, t] := SR[s_{j-1}, t']$, where t' is the nearest time in the future for which a departing event exists. Having this information for every j, the decoupled reliability of J is then simply $SR[s_0, t_D]$.

We can compute $SR[s_{j-1}, t]$ in the order of decreasing values of j. We initially set $SR[s_k, t_A] = 1$ (denoting that the fraction of successful sub-journeys arriving in s_k is 1, if the sub-journey starts in s_k and before t_A). The dynamic-programming like fashion for computing $SR[s_{j-1}, t]$ at any time t then follows from the following recurrence:

$$SR[s_{j-1}, t] = \frac{1}{m} \sum_{i=1}^{m} SR[s_j, t_i], \tag{5}$$

where t_i is the earliest arrival time of line l_j at stop s_j if the line uses timetable T_i and does not depart before time t from s_{j-1}.

When implementing the algorithm, we can save the (otherwise linear) time computation of the values of t_i from the recurrence by simply storing this value and updating if needed.

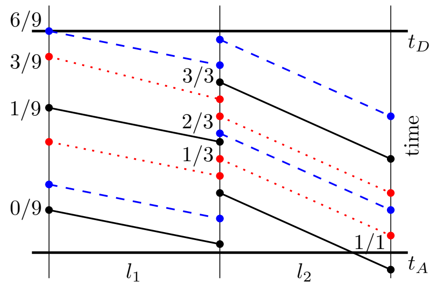

Figure 4 A journey with two lines l_1 and l_2 and three timetables (solid black, dotted red, dashed blue). The fractions denote the stored values of $SR[s_j, t]$.

Figure 4 illustrates the algorithm, and the resulting decoupled reliability of 6/9. The running time of a naive implementation is $\mathcal{O}(k \cdot (m + e \log e))$, where e is the maximum number of considered tram departing events at any station s_j.

6 Small Experimental Evaluation

In this section we describe and comment on a small experimental evaluation of the proposed approach to robust routing in public transportation networks. We first describe few observations/properties of our approach that serve as a kind of "mental" experiment. We have also implemented the proposed algorithms, and we report on our preliminary experiments with *real* public networks and *artificially* generated delays.

Properties of the approach Let T_1 and T_2 be two recorded timetables (from which we want to learn how to travel from stop a to stop b and arrive there before t_A). Consider the situation where the best journey J to travel from a to b in timetable T_1 is the same as the best journey to travel from a to b in timetable T_2. Assuming that T_1 and T_2 represent typical delays, common sense dictates to use the very same journey J also in the future. This is exactly what our approach does as well. Recall that $S_\gamma \leq 1$. Let r be the route that corresponds to the journey J. In our case, setting γ so that $A_\gamma(T_1) = A_\gamma(T_2) = \{r\}$, we get that $S_\gamma = \frac{|A_\gamma(T_1) \cap A_\gamma(T_2)|}{|A_\gamma(T_1)||A_\gamma(T_2)|} = 1$, and thus our approach computes the very same γ and returns the journey J as the recommendation to the user. These considerations can be generalized to the cases such as the one where $A_\gamma(T_1) = \{r\}$, $r \in A_\gamma(T_2)$, in which again J will be returned as the recommendation to the user.

If only a reliable journey is required, and the travel time is not an issue, then suggesting to depart few days before t_A is certainly sufficient. We now demonstrate that our approach does not work along these lines, and that it in fact reasonably balances the two goals *robustness* and *travel time*. We consider the symmetric situation where both $|A_\gamma(T_1)|$ and $|A_\gamma(T_2)|$ grow with γ in the same way, i.e., for every γ, $|A_\gamma(T_1)| = |A_\gamma(T_2)|$. Let us only consider discrete values of γ, and let γ_1 be the largest γ for which $A_{\gamma_1}(T_1) \cap A_{\gamma_1}(T_2) = \emptyset$. Let $x = |A_{\gamma_1}(T_1)|$. Then, for every $\gamma > \gamma_1$, $S_\gamma = \frac{\Delta_\gamma}{(x+\Delta_\gamma)^2}$ for some values of Δ_γ. Simple calculation shows that S_γ is maximized for $\Delta_\gamma = x$. We can interpret x as the number of failed routes (that would otherwise make it if no delays appear). Then, S_γ is maximized at the point that allows for another $\Delta_\gamma = x$ routes to joint the approximation sets $A_\gamma(T_i)$. Thus, the more disturbed the timetables are, the more "backward" in time we need to search for a robust route.

Experimental evaluation We implemented the algorithms presented in the sections 2, 3 and 4 in Java 7. The experiments were performed on one core of an Intel Core i5-3470 CPU

Table 1 Comparison of the described methods over 100 test cases.

	on time	less than 5 min late	less than 10 min late	avg arrival time	avg earlier depart. than Opt in T_3
Unexpected Similarity, pick u.a.r.	88%	95%	97%	7:54	3.14
Unexp. Sim., pick max. # occurences	89%	94%	97%	7:54	3.22
Optimum in T	31%	48%	60%	8:07	-7.9
2nd Optimum in T	49%	64%	76%	7:57	2.14
Opt. in T + end buffer time	41%	57%	70%	8:03	-3.26
Buffer time 3 min	55%	71%	83%	7:59	0.02
Buffer time 5 min	66%	81%	88%	7:56	4.43

clocked at 3.2 GHz with 4 GB of RAM running Debian Linux 7.0. We used the combined tram and bus network of Zurich as input. It has 611 stops and 90 different line IDs. In our experiments, the actual number of lines itself is much higher (471), since multiple lines may operate under the same ID (e.g., lines in opposite directions, or lines coming from or returning to the depot). The planned timetable T that we used is the official one for the Zurich network. However, trips departing before 6 a.m. or after 10 p.m. were ignored (since the timetable is only valid for 24 hours, trips starting before and ending after midnight are virtually interrupted at midnight, leading to a large number of lines).

We set the latest arrival time t_A to 8 a.m., and carefully chose a small set of problematic stops S' where delays usually occur. Then we generated 100 pairs of stops (a, b) uniformly at random. For each pair, we generated three timetables T_1, T_2 and T_3 from T by delaying every trip τ in T between 0 and 3 minutes at every station $s \in S'$ (if s occurs on τ). These delays are 0 or 3 minutes with probability 1/8, and 1 or 2 minutes with probability 3/8. T_1 and T_2 are used as input to the algorithm, and the arrival time of the computed journey is measured in T_3. We use the following methods for computing the journey.

1. **Maximizing the Unexpected Similarity** Compute a route using the approach described in section 4. We consider two ways to pick a route from the intersection: 1) choose uniformly at random; 2) Choose the one with the maximum number of occurrences.
2. **Optimum in T** Find the best or the second best journey according to the planned timetable T. Compute also the latest journey arriving in T five minutes before t_A.
3. **Buffer time for transfers** Consider the latest journey from a to b that arrives on time in T such that at each transfer stop it have to wait for an additional "buffer time". We experiment with buffer times of $1 - 5$ minutes.

For each of these statistics, we computed the following numbers (see Table 1): Percentage of the experiments where the proposed journey arrives on time, how often it arrives at most 5 minutes late, and how often it arrives at most 10 minutes late. We also computed the average arrival time of the journeys proposed by each method as well as the average difference between the departure time of the proposed journey to the optimal journey in T_3.

The average time for computing the optimum solution is 127ms, the time to compute a robust journey by using Unexpected Similarity is 262ms. We observed that our algorithm produces journeys that are on time in high percentage of cases, and on average we propose to depart only around 3 minutes earlier than the optimum in T_3, thus the cost we pay for this robustness is quite low. In comparison, the other considered approaches achieve much lower success rates. Even the generous buffer time of 5 minutes turns out not to be enough to beat our approach, which is rather surprising given the small delays in the considered timetables.

7 Discussion

We presented a novel framework for robust routing in frequent and dense urban public transportation networks based on observations of past traffic data. We introduced a new concept to describe a travel plan, a *journey*, that is not only well suited for our robustness issues, but also represents a natural and convenient description for the traveler. We also provided a bag of algorithmic tools to handle this concept, tailored towards the proposed robustness measures. We described a simple way to assess the reliability of a given journey. We also used a different approach to robustness and described how to find a robust journey according to it. We are preparing further experiments to confirm efficiency of the presented algorithms and to evaluate the quality of the computed robust journeys.

Future work is to examine how the described methods can be extended to support a fully multi-modal scenario, e.g., how to integrate walking. We believe that the modelling itself is easy, while the performance of the algorithms will decrease significantly unless we develop special techniques. Also considering and exploring different robustness concepts for journeys may be worthwhile.

References

1 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In Mark Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010*, volume 6346 of *LNCS*, pages 290–301. Springer Berlin Heidelberg, 2010.

2 Reinhard Bauer, Daniel Delling, and Dorothea Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, 2011.

3 Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 895–902. Society for Industrial and Applied Mathematics, 2001.

4 Joachim M. Buhmann, Matúš Mihalák, Rastislav Šrámek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In Robert D. Kleinberg, editor, *ITCS*, pages 505–514. ACM, 2013.

5 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering time-expanded graphs for faster timetable information. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 182–206. Springer Berlin Heidelberg, 2009.

6 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based public transit routing. *Algorithm Engineering and Experiments (ALENEX)*, pages 130–140, 2012.

7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *SEA*, volume 7933 of *LNCS*, pages 43–54. Springer, 2013.

8 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Experimental Algorithms*, pages 347–361. Springer, 2008.

9 H Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

10 Marc Goerigk, Martin Knoth, Matthias Müller-Hannemann, Marie Schmidt, and Anita Schöbel. The price of robustness in timetable information. In *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, pages 76–87, 2011.

11 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis,

editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer, 2009.

12 Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, pages 67–90. Springer, 2007.

13 Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Algorithms–ESA 2006*, pages 552–563. Springer, 2006.

14 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 12:2–4, 2008.