Routing in Stochastic Public Transit Networks

Barbara Geissmann
Department of Computer Science, ETH Zurich, Switzerland
barbara.geissmann@inf.ethz.ch

Lukas Gianinazzi
Department of Computer Science, ETH Zurich, Switzerland
lukas.gianinazzi@inf.ethz.ch

Abstract

We present robust, adaptive routing policies for time-varying networks (temporal graphs) in the presence of random edge-failures. Such a policy answers the following question: *How can a traveler navigate a time-varying network where edges fail randomly in order to maximize the traveler's preference with respect to the arrival time?* Our routing policy is computable in near-linear time in the number of edges in the network (for the case when the edges fail independently of each other).

Using our robust routing policy, we show how to travel in a public transit network where the vehicles experience delays. To validate our approach, we present experiments using real-world delay data from the public transit network of the city of Zurich. Our experiments show that we obtain significantly improved outcomes compared to a purely schedule-based policy: The traveler is on time 5-11 percentage points more often for most destinations and 20-40 percentage points more often for certain remote destinations. Our implementation shows that the approach is fast enough for real-time usage. It computes a policy for 1-hour long journeys in around 0.1 seconds.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Route Planning, Public Transit Network, Temporal Graphs

Digital Object Identifier 10.4230/OASIcs.ATMOS.2019.4

1 Introduction

Real-World networks, such as communication and transportation networks, change over time and can be subject to delays and edge failures. Routing a traveler (or packet) through such a stochastic and time-dependent network (from a source vertex to a destination vertex) is challenging and requires *robust routing policies* that not only recommend a fixed route, but give alternatives in case the intended route becomes infeasible.

What constitutes a "best" policy depends on the preference the traveler has with respect to the arrival time. For example, one goal could be to minimize the expected time of arrival. Another goal could be to maximize the probability to arrive before a given deadline. These goals are distinct in the stochastic setting because they express different attitudes towards risk. In particular, when the goal is to arrive before a given deadline, as the available time to reach the destination becomes smaller, the best policy might have to become more and more risky (i.e. choose edges that are more likely to fail) in order to have a chance to arrive on time. In contrast, if only the expected arrival time has to be minimized, a route is chosen that is "good in most cases". These preferences of the traveler can be expressed as *utility functions* on the arrival times.

Inherently time-dependent networks can be well-represented by *temporal graphs* [12, 1, 10], which model time explicitly: every edge is only available at certain points in time (consider Figure 1). For example, the connections of a public transit network are only available at certain points in time (when a vehicle departs).

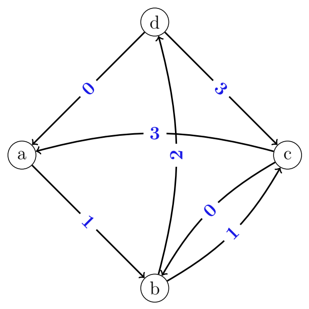

Figure 1 Every edge in a temporal graph can only be traversed at a certain ***availability time***. In the example, there is an edge from vertex b to vertex d at availability time 2.

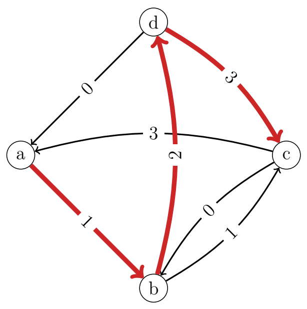

Figure 2 A ***strictly time-respecting path*** has increasing edge availability times. The bold path is the only strictly time-respecting path from a to c and has *arrival time* 3.

A core characteristic of traveling inside a public transit network is the possibility of missing a *transfer connection.* This can occur because of delays of a vehicle or even because the connecting vehicle leaves too early. When a connection breaks, the traveler needs to change their route during the journey. Therefore, small delays of a vehicle can cause larger delays for a traveler's journey.

This motivates adding *random edge failures* to a temporal graph, and thus obtaining a *faulty temporal graph.* For this setting, we show how to provide *optimal robust routing policies* for *any* efficiently computable *utility function* in near-linear time in the size of the temporal graph. In this paper, we show how public transit networks can be modeled as faulty temporal graphs and we give an algorithm for robust routing in public transit networks. We validate our robust routing policies by using *real-world* public transportation data. Note that our focus is not on catastrophic network failures due to accidents or other highly disruptive events, but on failures due to *everyday delays* caused for example by traffic congestion. In principle, our approach could also be applied to other time-varying and failure-prone networks such as ad-hoc or mobile phone networks.

1.1 Preliminaries

Temporal Graphs

To represent a network where edges can only be taken at a certain moment in time, we can use a temporal graph [12, 1, 10, 20, 8]. Formally, a temporal graph $G = (V, E, T)$ has vertices V and temporal edges E, where E is a multiset of directed edges on V. Each temporal edge $e = (u, v)$ has a nonnegative integer availability time $T(e)$. The semantic of the temporal graph is that at time $T(e)$, the edge $e = (u, v)$ can be used to go from vertex u to vertex v (see Figure 1). Note that there can be multiple temporal edges going from u to v. The number of edges that are incident to a vertex v is the degree $deg(v)$ of v.

Note that in a variant of temporal graphs (so-called interval graphs [10]) every edge is available during an *interval of time.* This is *not appropriate* for our purposes, as in our main application (for public transport), *connections are only available at discrete points in time.*

A *strictly time-respecting path* $p = e_1, \ldots, e_k$ in a temporal graph (V, E, T) is a path in the graph (V, E) where the edges have increasing availability times (according to T). That is, if two edges e_i and e_{i+1} follow each other in the strictly time-respecting path p, then $T(e_i) < T(e_{i+1})$ (see Figure 2). We call the availability time of the last edge e_k in a time-respecting path p the *arrival time* of the path p.

Faulty Temporal Graphs

When the connections of a network change probabilistically over time, we can model this using a temporal graph with edges that fail at random. Formally, we augment the definition of a temporal graph with a failure distribution F over the edges.

A *faulty temporal graph* $\mathcal{G} = (V, E, T, F)$ has vertices V, directed edges from a multiset E, discrete nonnegative edge availability times $T : E \mapsto \mathbb{N}$, and a failure distribution F. The number of vertices is $n = |V|$ and the (maximum) number of edges in the faulty temporal graph is $m = |E|$. A faulty temporal graph defines a random variable whose outcomes are temporal graphs with vertices V, edges that are a subset of E, and with availability times given by T. Specifically, for an edge $e \in E$ we say it is *potentially available* at the fixed *availability time* $T(e)$. The edge fails with probability p_e and $F(e)$ is the indicator random variable for the event that edge e fails. If not stated otherwise, we assume that the edges fail independently of each other. This assumption is relaxed in Section 2.3, where the edge distributions follow a kind of Markovian assumption.

Robust Adaptive Routing

In the *robust adaptive routing* problem, a traveler in a faulty temporal graph starts out at a designated starting vertex at time 0. Whenever the traveler arrives at a vertex i at a time t, the traveler picks a temporal edge $e = (i, j)$ with availability time $T(e)$ larger than t. At that time $T(e)$, the traveler tries to go across this edge. If the edge does not fail, the traveler succeeds and arrives at the endpoint j of that edge at time $T(e)$. If the edge fails, the traveler remains at vertex i and must pick a new edge to take with availability time larger than $T(e)$. Note that this means that the traveler traverses a strictly time-respecting path in the faulty temporal graph using only edges that did not fail. The goal of the traveler is to maximize the expectation of a computable *utility function* of the time at which a destination vertex is reached. For example, they might want to arrive at a destination vertex before the *deadline* x with the largest possible probability. Then, the utility is 1 if the traveler arrives on time and 0 otherwise.

The algorithmic question that solves the robust adaptive routing problem is to preprocess the faulty temporal graph such that we can quickly answer the following *routing query*:

"When arriving at vertex i at time t, where should the traveler go next?"

A set of answers to these routing queries is called a *routing policy*. We are interested in *optimal routing policies* in the sense that they maximize the *expected utility* of the traveler.

If desired for a certain application, a routing policy could also be used to generate a temporal path a-priori (such a path represents the journey in case no connection breaks and can be thought of as an optimistic preview). This path would be followed until one of the edges fails. In this event, the policy would be queried again to compute a new path.

We continue with a more formal statement of the routing problem. We are given a faulty temporal graph $\mathcal{G} = (V, E, T, F)$ and a set of destination vertices $S \subseteq V$. A *policy* P maps every (non-destination) vertex $i \in V - S$ and every time t to an edge $e' = (i, j)$ with larger availability time $T(e') > t$, or to a special symbol $\perp$ in case no edge $e' = (i, j)$ with availability time larger than t exists. The semantics of the policy are such that if the traveler is at vertex i at time t, they choose the edge $e = P(i, t)$ to traverse next according to the policy. If the edge $e = (i, j)$ does not fail, the traveler goes to the other endpoint j of e at time $T(e)$ and continues to choose an edge from there. Otherwise, the traveler stays at vertex i but the time also changes to $T(e)$. The traveler *stops* as soon as they reach a destination vertex or once the policy returns $\perp$, which means that the traveler is *stuck*.

We formalize the preference of the traveler with respect to the arrival time in a *utility function.* Such a *utility function* $U_i(t)$ maps every vertex $i \in S$ and every time t to a real-valued utility, where higher values correspond to a higher preference of the traveler. This terminology highlights the relation to Stochastic Optimal Control [2], where an agent tries to make decisions that optimize their expected utility. The idea is that if we arrive at vertex i through an edge e with availability time $t = T(e)$, the utility for the traveler is $U_i(t)$. For example, to maximize the *on-time arrival probability*, the utility is 1 if the traveler arrives at a destination vertex before a given deadline and 0 otherwise. If the traveler ever gets stuck, the utility obtains a smallest possible value that we denote by U^0. For example, if the utility corresponds to the probability to arrive at a destination on time, then $U^0 = 0$.

The *utility of a policy* P starting from starting vertex i and starting time t is the value of the utility function at the vertex and time where the traveler stops. Note that the utility of a policy is a random variable. We consider the *expected utility* of the policy P, where the expectation is over the random failures of the edges of the faulty temporal graph.

1.2 Related Work

There is a vast variety of approaches to path finding problems in stochastic networks. We can categorize approaches based on the following criteria:

- **A-priori or Adaptive.** Does the traveler decide upfront which way to go (*a-priori*) [7, 15, 17] or can they change the route along the way (*adaptive*)?
- **Time-dependent or Time-independent.** Does the network change with time (*time-dependent*) or not (*time-independent*)? In a time-dependent network, the time it takes to go between two vertices changes depending on the time the traveler attempts to do so. An extreme case is when certain links are only available at discrete points in time.
- **Scoring Criterion.** How are different outcomes scored for the traveler? For example, does the traveler of the *stochastic* network want to maximize the *on time arrival* probability (SOTA) or does the traveler want to obtain a *least expected arrival time* (LET). Note that a utility function is not the only way to score a path. Alternatives include approaches which search for paths that are *pareto-optimal* with respect to multiple criteria [5, 16].
- **Runtime.** Is the solution obtained in polynomial time, pseudo-polynomial time (i.e. it depends on the number of time steps in the problem), or super-polynomial time?

Adaptive and Time-Independent

There are two motivations to take an adaptive approach as opposed to an a-priori approach. First, a-priori probabilistic path problems have only been solved in polynomial time for special cases (like for affine and exponential utility functions [15]) and hence it is pragmatic to take a different approach. Second, the outcome for the traveler can be improved if "live" information can be incorporated into the decision making process.

Fan and Nie [6] show termination for an algorithm to solve the adaptive SOTA problem (in the continuous time domain). They propose a set of (integral) equations which are solved iteratively, starting out with a trivial approximation, then using the approximation of the last iteration to compute the next iteration of the utility functions. Although they show convergence of the approximation to the true value, the algorithm can take an exponential number of iterations.

Samarayake et al. [19] observe that there is a minimum time that it takes to traverse an edge. They obtain pseudo-polynomial runtime (in the number of such traversals that can occur within the time-budget). Instead of computing the utility functions iteratively, they construct parts of the functions one after the other. They discretize the problem by dividing the time-domain into y evenly sized pieces (y is a parameter such that the pieces are smaller than the minimum time it takes to traverse an edge). On this discretized version, they obtain a runtime of a $O(my^2)$, where m is the number of edges.

Because the SOTA utility functions are not of some nice form that can be integrated efficiently, the surveyed approaches all eventually discretize the time-domain. Samarayake et al. use evenly spaced time-intervals and do not provide bounds on the approximation quality as a function of the number of time-interval. This downside is addressed by Hoy and Nikolova [11], who give a polynomial-time approximation scheme for the SOTA and LET problem on *acyclic directed graphs* that obtains an additive error of $1/\epsilon$ in $O(mn^2/\epsilon^2)$ time. Similar to our approach, they can handle general scoring functions that depend on the arrival time of the traveler.

Adaptive and Time-Dependent SOTA

Transit networks consisting of trains and buses are different from street networks because vehicles that connect physical locations in a transit network are only available at a specific point in time (before the vehicle leaves a station), whereas streets remain available most of the time (although delays and infrequent disruptions are possible). In particular, missed transfer connections between distinct vehicles can play a crucial role in transit networks.

Keyhani et al. [13] looks at estimating the reliability of transfers and fixed (a-priori) paths in a train network. Keyhani [14] deepens this work on the reliability of train transfers and connections. They present an adaptive approach to solving SOTA in a similar transit network problem setting as ours. However, they allow the traveler to change the route depending on the arrival time at every vertex. This requires a model of the arrival-time distributions and leads their algorithms to have pseudo-polynomial runtime in the size of the support of the arrival-time distributions. Another difference to our work is that Keyhani does not represent the schedule as a temporal graph, but use their own problem-specific model.

Adaptive and Time-Dependent LET

In the *bus network problem* [3], the traveler decides whether to take a bus whenever it arrives. The traveler has access to the statistics of the bus arrivals, but they do not know exactly when a bus will actually arrive, until it arrives. The goal is to reduce the *expected time* that a policy takes to move a traveler from the start station to the destination station.

Boyan and Mitzenmacher [3] present results for the case when the buses arrive independently of each other and satisfy additional conditions (in particular they can be distributed according to exponential, uniform, or normal distributions). They generalized a previous more limited result by Datar and Ranade [4]. In order to compute a policy minimizing the expected travel time in polynomial time, they need to be able to compute the expected arrival time of a bus given that it has not arrived yet. However, exact computation of these expectations involves a convolution that can take polynomial time in the number of time steps considered and it is not shown how an approximate solution to the expectations impacts the accuracy of the result.

In contrast to our problem setting, the traveler in the bus network problem can change their decision whenever a bus arrives (whereas Keyhani [14] and in our model we only allow a decision when the traveler arrives at a station or a connection breaks). On the other hand, in our model we allow more general delay distributions and *our runtime is strongly polynomial*.

1.3 Our Contribution

We obtain a near-linear runtime for computing an optimal robust adaptive routing policy in a faulty temporal graph. For a faulty temporal graph with m edges that fail independently of each other, computing an optimal routing policy takes $O(m \log m)$ time. We need $O(m)$ space to store the policy and take $O(\log m)$ time per query. We allow any utility function that can be evaluated in $O(\log m)$ time.

When the edge failure probability depends on the last edge the traveler attempted to traverse, we compute an optimal routing policy in $O(m \log m + \sum_{i \in V} \deg^2(i))$ time.

As an application of our model, we represent traveling inside a *public transit network that is subject to delays* as robust adaptive routing in a faulty temporal graph. We transform a timetable with N entries where at most d vehicles run through any station into a faulty temporal graph with $O(Nd)$ edges. This gives $O(Nd \log N)$ time to compute a robust routing policy for a transit network with independent delays. We evaluate our routing policy using real-world transit network delay data from the public transit network of the city of Zurich. We compare our approach to a traveler that travels to arrive as early as possible using only the schedule provided by the city of Zurich and to a traveler that has perfect knowledge of all future delays. Our evaluation shows that our model is *accurate in predicting the probability of being on time* and our routing policy provides (in less than 0.1 seconds) significant improvements over an approach that neglects delays.

2 Robust Adaptive Routing

The efficiency and generality of our approach is enabled by two observations. First, the problem has an acyclic nature, since the traveler navigates strictly time-respecting paths. Hence, a dynamic programming formulation emerges. This initial dynamic program (presented in Section 2.1) is, however, too slow because it depends on the largest availability time. Second, only certain points in time matter (those where there is an edge with that availability time). This leads to an improved dynamic program (described in Section 2.2) that achieves near-linear runtime.

2.1 Pseudo-Polynomial Time Algorithm

We start out with a basic dynamic program to compute a routing policy for any faulty temporal graph (with independent edge failures).

For every vertex $i \in V$ and every time $t \in \mathbb{N}$, we denote the computed expected utility starting from vertex i at time t with $u_i(t)$. The basic idea is to find a recursion for $u_i(t)$, parameterized by the current vertex i and the current time t. Since the traveler traverses a strictly time-respecting temporal path, the traversed edges have increasing availability time. Therefore, the subproblems overlap in an acyclic way and this gives a dynamic program.

Let us start with the base cases. If the vertex i is a destination vertex, the expected utility $u_i(t)$ coincides with the value of the utility function, hence we set $u_i(t) = U_i(t)$.

Next, we recursively describe the best decision to take being in vertex i at time t. Intuitively, the idea is to try every incident outgoing edge and then take the best such edge. For this, we need to compute the expected utility given that we take a particular edge. For each such edge e with an availability time $T(e)$ larger than the current time t, we condition on the event that the edge e fails. Using the law of total expectation, we relate the expected utility at time t to an expected utility at some time larger than t. If an edge $e = (i, j)$ fails, the traveler stays at vertex i and the (conditional) expected utility is $u_i(T(e))$. Otherwise, the traveler reaches vertex j and the (conditional) expected utility is $u_j(T(e))$. We can express this in the following recursive formula (for any vertex i that is not a destination):

$$u_i(t) = \max_{\substack{e=(i,j)\in E \\ T(e)>t}} \left((1-p_e)\cdot u_j\Big(T(e)\Big) \quad + \quad p_e \cdot u_i\Big(T(e)\Big) \right) \ .$$

For any time t and vertex i, the expected utility $u_i(t)$ at vertex i and time t only depends on values $u_j(t')$ with $t' > t$. Hence, we can process the expected utilities $u_i(t)$ in order of decreasing time t. When the traveler is at some vertex i at some time t, the routing policy is to take the edge $e = (i, j)$ which obtains the maximum value in the expression for $u_i(t)$ (take any edge if several edges are tied for the same value). If there is no edge e with $T(e) > t$, then $u_i(t) = U^0$. In that case, the traveler is stuck and cannot reach a destination at all (i.e., the policy returns $\perp$). See Appendix A.1 for an inductive correctness proof of the algorithm.

Let x be the largest availability time that occurs in T, then the runtime of this approach is $O(m\ x)$. Initializing the base cases takes time $O(x + n)$. Afterwards, each of the vertices needs to compute at most x different values (the utility for all times larger than x is trivially U^0 and does not need to be computed). To compute the value for a particular vertex and time, we need to look up an already computed value for each of the neighbors. Thus, a vertex i with degree $\deg(i)$ takes $O(\deg(i)\ x)$ time to find its best decision. The runtime is thus $O(\sum_{i\in V} \deg(i)\ x) = O(m\ x)$. As we explicitly store the result to all routing queries, the routing policy uses $O(n\ x)$ space and each routing query takes $O(1)$ time.

2.2 Near-Linear Time Algorithm

The problem with the basic dynamic program is that its runtime and space depends on the largest availability time x. This value is not polynomial in the input size and so the basic dynamic program runs in pseudo-polynomial time. In practice, this means that increasing the time-resolution of the data (say from measuring in minutes to seconds) also increases the runtime of the algorithm proportionally. We now show how to reduce the runtime of the basic algorithm by improving the order in which we evaluate the recursion and by leaving out redundant points in time. The new algorithm runs in near-linear time.

Observe that only those times are relevant for the traveler where there is some incident edge. More precisely, if there is no edge *leaving* vertex i inside some time interval $t_1, \dots, t_2$, then the expected utility for vertex i is the same for all those times. This is because when the traveler is at vertex i, the traveler cannot take any new decision in that time range and the traveler does not learn any new information. It thus suffices to compute the expected utility for t_2. We store the computed utilities sorted by increasing times for each vertex. To query the expected utility at a certain time, we do a binary search for the next largest time that has a computed value.

A closer look at the recursive equation reveals that the expected utility for vertex i and time t is the maximum of the expected utility at time $t + 1$ and the maximum possible expected utility given that we take an edge leaving at time $t + 1$. We therefore process the *edges* in decreasing order of availability times. For each edge $e = (i, j)$, we compute the best possible expected utility $u(e)$ given that the traveler plans to take this edge e using the expression

$$u(e) = (1-p_e)\cdot u_j\Big(T(e)\Big) \quad + \quad p_e \cdot u_i\Big(T(e)\Big) \ .$$

After processing all edges at time $t+1$, we update all adjacent vertices. For each vertex i that has some edge leaving at time $t+1$, we set

$$u_i(t) = \max\left(\left(\max_{\substack{e=(i,j)\in E\\ T(e)=t+1}} u(e)\right), u_i(t+1)\right) .$$

If the maximum expected utility is obtained by taking some edge e at time $t+1$, this edge is chosen for time t. Otherwise, the same edge is chosen as for time $t+1$. Recall that we do not explicitly store the choices and utilities for all times and vertices, but only remember decisions for vertices and times, where the vertex has an outgoing edge.

▶ **Theorem 1.** *Computing an optimal routing policy in a faulty temporal graph with independent edge failures takes $O(m \log m)$ time. The policy uses $O(m)$ space and a routing query takes $O(\log m)$ time.*

Proof. Sorting the edges takes $O(m \log m)$ time. Then, each edge e is processed once to compute $u(e)$, which uses two already computed utilities. Looking up those utilities takes $O(\log m)$ time (by using a binary search for the successor). Updating a vertex at a certain time takes time proportional to the number of edges at that time, so $O(1)$ per edge. Inserting a new expected utility value into the sorted array takes $O(1)$ amortized time by using standard array doubling (store the array in decreasing order of time so that inserting a new expected utility always occurs at the end of the array). ◀

2.3 Last-Edge Markovian Failures

So far, we assumed the edges to fail completely independently of each other and independently of time. We can also consider the situation when the probability for an edge to fail depends on the last edge the traveler planned to take (they either attempted to traverse this edge and failed or succeeded to traverse this edge). By replacing our independence assumption by a Markovian independence assumption given the last edge the traveler attempted to take, we obtain *Last-Edge-Markovian failures*. As detailed in Appendix B, we can modify our dynamic program to obtain the following result:

▶ **Theorem 2.** *Computing an optimal routing policy in a faulty temporal graph with Last-Edge-Markovian failures takes $O(m \log m + \sum_{i\in V} deg^2(i))$ time. The policy uses $O(\sum_{i\in V} deg(i))$ space and a routing query takes $O(1)$ time.*

3 Applications in Public Transit Networks

Traveling in a public transit network in the presence of delays can be modeled as robust adaptive routing in a faulty temporal graph, as we show in this section. In a public transit network, there are several *lines* of buses, trains, trams, and other vehicles. Each of those lines connects a series of *stations* in a predetermined order. Along each line, vehicles run according to a *schedule* which prescribes when a vehicle is supposed to arrive and to leave a station. The schedule contains N tuples that contain the line of the vehicle, the departure time, departure station, arrival time, and arrival station.

We assume that the traveler leaves the start station s_{start} at the starting time and wants to arrive at the destination station s_{dest} while maximizing their preference with respect to the arrival time: this preference is expressed as a *utility function* $u(t)$ of the arrival time and the goal is to maximize the expected value of this utility function (the *expected utility*).

At every station, the traveler can get off the current vehicle and attempt to *transfer* onto another vehicle (which succeeds if the latter vehicle departs *after* the arrival time of the current one plus the time it takes to transfer between the two vehicles).

Our model restricts traveling to routes that are feasible according to the schedule. This is somewhat pessimistic in that certain connections infeasible in the schedule could be feasible in practice due to delays and early arrivals. However, we argue that being slightly pessimistic is compatible with our goal of giving robust routes. Moreover, recommending such infeasible routes might be counter-intuitive to users of a transit system.

3.1 Public Transit Network Model

Temporal Graph Model

We map the transit network onto a temporal graph $G = (V, E, T, F)$. Note that our model is related to the time-expanded-graph model in [18], where each edge has a weight instead of an availability time. Without modeling failure probabilities, the latter model can be used to compute a route which minimizes the earliest arrival time.

To construct the temporal graph, we first add the vertices and edges that correspond to a single-hop ride with a vehicle. Say that, according to the schedule, some vehicle of line l leaves station s at time t and arrives at station s' at time t' . Then, there is a *departure vertex* (dep, l, s, t) and an *arrival vertex* (arr, l, s', t') connected by a temporal edge with availability time t'. Next, we add the connections that correspond either to transfers or to staying in a vehicle. In particular, there is an edge connecting every arrival vertex (arr, l, s', t') to every departure vertex $(\text{dep}, l', s', t'')$ at the same station s' with larger departure time than arrival time $(t'' > t')$. Finally, there is a special extra vertex *start*. The *start* vertex is connected to every departure vertex $(\text{dep}, l, s_{\text{start}}, t)$ at the start station s_{start} with a temporal edge at time $t - 1$ equal to the starting time minus 1. This shifting by one is necessary since the traveler traverses strictly time-respecting paths.

The temporal graph has $n = 2N + 1$ vertices (recall that N is the size of the schedule). Because of the transfer edges, the number of edges of the constructed graph depends on the largest number of vehicles that pass through a station. Let d be this maximum number of vehicles per station. Then, the temporal graph contains $m = O(Nd)$ edges.

Every arrival vertex $(\text{arr}, l, s_{\text{dest}}, t)$ at the destination station s_{dest} is a destination vertex. The *utility at* such *a destination vertex* $(\text{arr}, l, s_{\text{dest}}, t)$ should correspond to an estimate of the expected utility when using the vehicle v that arrives at that vertex. Given a list of observed arrival times $t_1, \ldots, t_k$ for vehicle v at station s_{dest}, compute $U_{s_{\text{dest}}} = \left(\sum_i^k U_{s_{\text{dest}}}(t_i)\right)/k$, the average value of the utility function for those arrival times (Justification in Appendix A.2).

We now describe how to set the *edge failure probabilities* based on the probability that vehicles are delayed. For simplicity, we assume that neither edges corresponding to traveling (these go from a departure vertex to an arrival vertex) nor edges that correspond to staying inside a vehicle can fail. This means that only transfer edges, which go from an arrival vertex of some line l to a departure vertex of some other line l' can fail completely. Intuitively, the probability for this edge to fail is the probability that we are too late to catch the connection. We are given samples for the arrival time of a vehicle a, departure time of vehicle b, and transfer time between the two platforms. Then, the failure probability for the transfer edge assuming independent vehicle travel is estimated as the fraction of samples where the transfer is infeasible (i.e. the arrival time plus transfer time is larger than the departure time).

Note that we do not require to model the actual delay of vehicles (which can be time-dependent and congestion-dependent [9]), since we are only interested in the probability to miss a connection.

Computation of the Policy

We can apply our policy construction algorithm from Section 2.2 to our model of a transit network. We call such a policy a *robust transit network routing policy* and we obtain the following bounds, which follow from Theorem 1 since the number of edges is in $O(Nd)$:

▶ **Corollary 3.** *For a schedule of size N where at most d vehicles run through any station, computing a robust transit network routing policy with independent edge failures takes $O(Nd \log N)$ time and uses $O(Nd)$ space.*

3.2 Experimental Methodology

We evaluate our algorithm by applying it to the public transportation network of the city of Zurich (ZVV network). We investigate the performance of our routing policy on real transit network delay data for the year 2018. Throughout all experiments, we consider a traveler who wants to arrive at a given destination station at a given (hard) *arrival deadline* time x, within some *time budget* b (i.e., the traveler starts the journey at time $x - b$ in some station). We considered three main questions for our study:

1. **Quality of the solution.** How well does the computed policy compare to a deterministic *schedule-based* policy and how much room for improvement is there compared to an *oracle policy* with perfect knowledge of the future?
2. **Model error.** How well do the predicted utilities (according to the policy) match the simulated utilities when following the policy? The model error evaluates the underlying model assumptions empirically (e.g., that edges fail independently).
3. **Runtime.** How does the time to compute a policy scale with the time budget b?

We evaluated these questions for time budgets between 10 and 60 minutes in 10-minute increments. For our policies, we train an edge failure model that uses the *last two weeks of delay data* before the first evaluated day.

Evaluation Approach

We evaluate the performance of the candidate policies for 38 destinations (and for each destination for every possible departure station) and 13 random destination deadlines between 7am and 6pm.

For the evaluation, we first compute the frequency (during a 1 month period) with which we can reach a given destination from a given source station at a given time using a given candidate policy (and we repeat this for every such set of parameters). Then, we compute the desired quantity (either an error value or some difference in utility) by averaging over all source stations for the given destination. For the error metrics we take the average over those source stations which reach the destination with nonzero probability (using the oracle policy). Note that for the other source stations the error metric is trivially 0.

In Figures 3 – 6 we use box plots to show the results, where the dot indicates the median of the values, the boxes indicate the lower and upper quartile, and the whiskers indicate minima and maxima. Note that each plot summarizes data for different destination stations and deadlines (and thus the variation is not due to probabilistic reasons only, but mainly due to differences depending on the destination stations). See Appendix C for a more detailed description of our experimental setup.

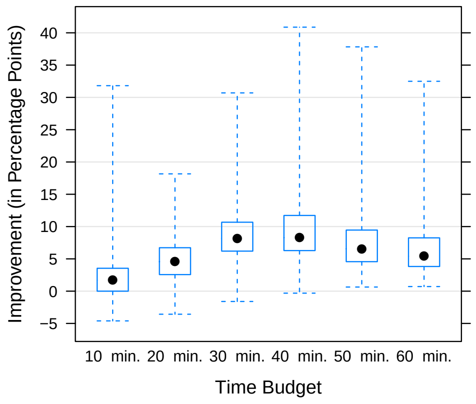

Figure 3 Average improvement over the schedule-based policy, plotted by time budget.

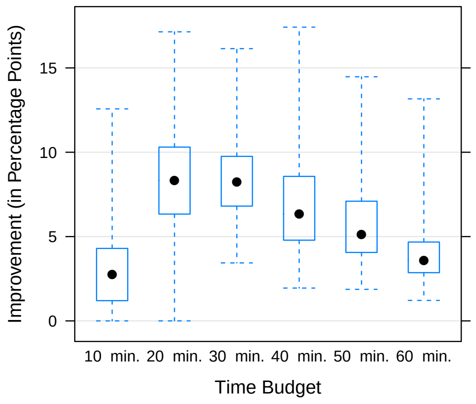

Figure 4 Average potential for improvement of our policy with respect to the oracle policy, by time budget.

3.3 Results

Quality of the Solution

Figure 3 shows that the largest improvement (with respect to the schedule-based policy) occurs for time budgets of 30 and 40 minutes, where it is $7-11$ percentage points depending on the destination station and deadline. This makes sense, as for too short time budgets there are few possible routes and there is not much to improve, while for very long time budgets, the choices matter less as the traveler has enough slack time for delays. For the longer configurations the median improvement is around 5 percentage points. For the very short configurations (i.e., 10 minutes) the median improvement is only slight with $1-2$ percentage points. Note that there are several destination stations where our improvement over the deterministic policy is especially high (i.e., 10-40 percentage points). These are the stations reachable by a single bus, where delays have a larger impact. Figure 4 shows that the maximum potential for improvement over our policy lies between 2 and 8 percentage points, where our policy is closer to optimal for longer time budgets. In conclusion, we see significant improvements for the vast majority of configurations. Improvements are on the order of improving the probability to be on time by around $5-11$ percentage points.

Model Accuracy

Figure 5 shows that the mean model error is small (less than 5 percentage points for all time budgets). With increasing time budget, the mean error decreases slightly. Note that the mean error is negative for most of the cases, which means that the simulated utility from the policy is better than the expected utility from the model. Since the mean model error is relatively small, this shows that in practice, *the assumption that the edges fail independently of each other does not significantly impact the applicability of the approach.*

Note that the obtained error is nontrivial, as, interestingly, using 5 months old delay data would yield a median error of 25 percentage points (and provide no improvement w.r.t. the schedule-based policy): Our approach is able to capture seasonal variations in delay distributions.

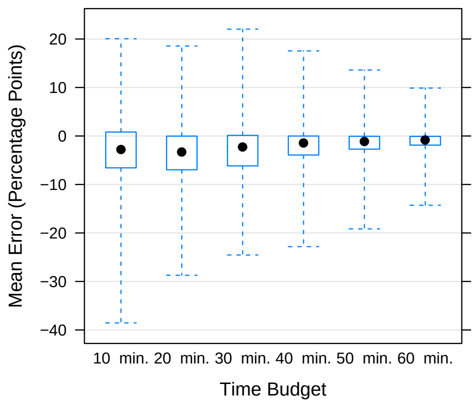

Figure 5 Mean difference between the computed utility according to the policy and the observed utility.

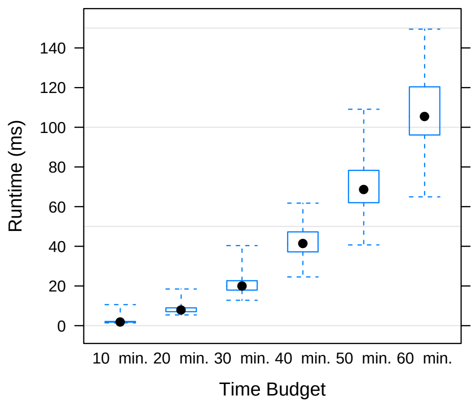

Figure 6 Time to compute a policy for varying time budgets.

Runtime

As seen in Figure 6, the median runtime is around 0.02 seconds for the 30 minutes time budget and around 0.105 seconds for the 60 minutes time budget. As the time budget b increases, we expect the runtime to grow slightly faster than proportional to b^2. This is because the number of possible transfers (and hence the size of the graph) increases quadratically with the time budget. The observed runtimes roughly follow the predicted trend.

4 Conclusion

We showed an approach to robust adaptive routing in time-dependent networks that both is tractable (computable in near-linear time in the size of the network) and yields useful improvements in practice over a purely schedule-based routing despite our simplifying assumptions (as exemplified by our analysis of travel inside a public transit network).

One next step could be to try variations on how to train the edge probabilities. It would be interesting to investigate other types of dependencies between the edge failure distributions and how they affect the quality of the solution.

We saw that the age of the training data affects the results. In principle, one could alter the model every day and always use the most up-to date delay data (say over the last one or two weeks) instead of changing the model every month as we did in our experiments. Another extension would be to include other forms of travel, for example travel by foot. This would not require a fundamental change in the model, but just a way to estimate travel durations for these trips.

In terms of theory, it would be interesting to know how the error in approximating the edge failure probabilities affects the error in the quality of the solution.

Future work could also include looking at applications of our approach for routing in *ephemeral or ad-hoc communication networks*. In that context, a distributed computation of the policy might be interesting.

References

1 Eleni C. Akrida, Jurek Czyzowicz, Leszek Gasieniec, Lukasz Kuszner, and Paul G. Spirakis. Temporal Flows in Temporal Networks. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 43–54, 2017. doi:10.1007/978-3-319-57586-5_5.

2 Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition.* Athena Scientific, 2005. URL: http://www.worldcat.org/oclc/314894080.

3 Justin A. Boyan and Michael Mitzenmacher. IMproved results for route planning in stochastic transportation. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 895–902, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365803.

4 Mayur Datar and Abhiram G. Ranade. Commuting with delay prone buses. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, pages 22–29, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338228.

5 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria Shortest Paths in Time-Dependent Train Networks. In *Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, pages 347–361, 2008. doi:10.1007/978-3-540-68552-4_26.

6 Yueyue Fan and Yu Nie. Optimal Routing for Maximizing the Travel Time Reliability. *Networks and Spatial Economics*, 6(3):333–344, September 2006. doi:10.1007/s11067-006-9287-6.

7 H. Frank. Shortest Paths in Probabilistic Graphs. *Operations Research*, 17(4):583–599, 1969. doi:10.1287/opre.17.4.583.

8 Viswanath Gunturi, Shashi Shekhar, and Arnab Bhattacharya. Minimum Spanning Tree on Spatio-Temporal Networks. In *Database and Expert Systems Applications, 21th International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part II*, pages 149–158, 2010. doi:10.1007/978-3-642-15251-1_11.

9 Dirk Heidemann. Queue length and delay distributions at traffic signals. *Transportation Research Part B: Methodological*, 28(5):377–389, 1994. doi:10.1016/0191-2615(94)90036-1.

10 Petter Holme and Jari Saramäki. Temporal Networks. *CoRR*, abs/1108.1780, 2011. arXiv:1108.1780.

11 Darrell Hoy and Evdokia Nikolova. Approximately Optimal Risk-averse Routing Policies via Adaptive Discretization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3533–3539. AAAI Press, 2015. URL: http://dl.acm.org/citation.cfm?id=2888116.2888207.

12 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 504–513, 2000. doi:10.1145/335305.335364.

13 Mohammad H Keyhani, Mathias Schnee, Karsten Weihe, and Hans-Peter Zorn. Reliability and delay distributions of train connections. In *OASIcs-OpenAccess Series in Informatics*, volume 25. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

14 Mohammad Hossein Keyhani. *Computing Highly Reliable Train Journeys.* PhD thesis, Technische Universität, Darmstadt, 2017.

15 Ronald Prescott Loui. Optimal Paths in Graphs with Stochastic or Multidimensional Weights. *Commun. ACM*, 26(9):670–676, September 1983. doi:10.1145/358172.358406.

16 Matthias Müller-Hannemann and Mathias Schnee. Finding All Attractive Train Connections by Multi-criteria Pareto Search. In *Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20-25, 2004, 4th International Workshop, ATMOS 2004, Bergen, Norway, September 16-17, 2004, Revised Selected Papers*, pages 246–263, 2004. doi:10.1007/978-3-540-74247-0_13.

17 Mehrdad Niknami and Samitha Samaranayake. Tractable Pathfinding for the Stochastic On-Time Arrival Problem. In *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, pages 231–245, 2016. doi:10.1007/978-3-319-38851-9_16.

18 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Experimental Comparison of Shortest Path Approaches for Timetable Information. In *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 88–99, 2004.

19 S. Samaranayake, S. Blandin, and A. Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C: Emerging Technologies*, 20(1):199–217, 2012. Special issue on Optimization in Public Transport+ISTT2011. doi:10.1016/j.trc.2011.05.009.

20 B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

A Correctness Proofs

We show that the computed policies indeed achieve the largest possible utility. Moreover, to justify our transit network model, we slightly generalize our notion of utility functions to *probabilistic utility functions*. Then, we show that computing an optimal policy with respect to the expectation of this probabilistic utility suffices to obtain an optimal policy with respect to the probabilistic utility function.

We denote the expectation of a random variable X as $\mathbf{E}[X]$, denote its conditional expectation given another random variable Y with $\mathbf{E}[X|Y]$, and denote the probability of an event E with $\mathbf{P}[E]$.

A.1 Deterministic Utility Functions

We show that the algorithm presented in Section 2.1 computes an optimal policy. The algorithm in Section 2.2 is equivalent, as already argued therein.

▶ **Theorem 4.** *The algorithm from Section 2.1 computes a policy that obtains the largest possible expected utility for all start vertexes and start times.*

Proof. The proof is by strong induction with decreasing time. The basic idea is that a policy with largest expected utility starting from vertex i at time t must try to use some edge e leaving i at time larger than t and use a policy that maximizes the expected utility for each of the two possible outcomes (edge e fails or does not fail).

Let $u_i^\star(t)$ be the largest possible expected utility of any policy starting at vertex i at time t. That is, $u_i^\star(t)$ gives the *true optimal utility*, whereas $u_i(t)$ is the *computed utility*. The proof consists of showing the two are equal.

For any time t, the induction hypothesis $H(t)$ is that for all times $t' > t$ and all vertexes i, we have that $u_i(t') = u_i^\star(t')$. Assume that $H(t)$ holds for some $t > 0$. We show that $H(t-1)$ holds. Consider some vertex i. If i is a destination vertex, then $u_i(t) = U_i(t) = u_i^\star(t)$ holds by construction. By $H(t)$, then $u_i(t') = U_i(t') = u_i^\star(t')$ holds for any $t' > t$.

Next, consider the case where i is not a destination vertex. If there is no edge leaving vertex i at a time larger than t, then $u_i(t) = U^0 = u_i^\star(t)$, as there is no way to reach the destination. Otherwise, the following equation is used to compute $u_i(t)$:

$$u_i(t) = \max_{\substack{e=(i,j)\in E\\ T(e)>t}} \left((1-p_e)\cdot u_j\Big(T(e)\Big) \quad + \quad p_e \cdot u_i\Big(T(e)\Big) \right) \ .$$

By induction hypothesis, all the utilities appearing on the right-hand side correspond to the largest possible expected utilities:

$$u_i(t) = \max_{\substack{e=(i,j)\in E\\ T(e)>t}} \left((1-p_e)\cdot u_j^\star\Big(T(e)\Big) \quad + \quad p_e \cdot u_i^\star\Big(T(e)\Big) \right) \ .$$

Let P_e be the policy that uses edge e first and continues using the optimal policy after that. Let $U(P_e)$ be its utility starting from vertex i at time t. Then, we can see that:

$$\begin{aligned} u_i(t) = \max_{\substack{e=(i,j)\in E\\ T(e)>t}} \Bigg(&\mathbf{P}[F(e)=1]\cdot \mathbf{E}[U(P_e) \mid F(e)=1] \\ &+ \ \mathbf{P}[F(e)=0]\cdot \mathbf{E}[U(P_e) \mid F(e)=0] \Bigg) \\ = \max_{\substack{e=(i,j)\in E\\ T(e)>t}} &\mathbf{E}[U(P_e)] \\ = u_i^\star(t) \ \ , \end{aligned}$$

where the last step follows because among all possible policies P_e we choose the one with largest expected utility. Finally, note that $H(t)$ implies that $u_i(t') = U_i(t') = u_i^\star(t')$ holds for any $t' > t$. ◀

A.2 Probabilistic Utility Functions

Recall that in our application to public transit networks in Section 3.1, we set the utility at a destination vertex to the average utility for the observed arrival times. We proceed to justify this as a way to compute optimal policies with respect to *probabilistic utility functions*.

For each destination vertex i, we introduce a *random variable* $\mathbb{U}_i$ that gives a utility distribution at the destination vertex i (given that we arrive at vertex i). In order to be able to define the expected value of a policy with respect to such utilities, we require that these random variables have finite expectation. Moreover, we assume that each random variable $\mathbb{U}_i$ is independent of the edge failure variables F. As before, we require that there is a smallest (deterministic) utility U^0 with the property that if the traveler gets stuck at a non-destination vertex j, their utility is always $\mathbb{U}_j = U^0$.

The *expected utility* of a policy with start vertex i and start time t is now the expected value of the utility function at the vertex where the traveler stops. Here, the expectation is both over the random edge failures and the outcome of the utility functions.

We prove that if we *replace the probabilistic utility functions with their expectations* and compute a policy with maximum value with respect to these deterministic utilities, we obtain a policy with maximum expected utility with respect to the probabilistic utilities.

▶ **Lemma 5.** *For every destination vertex i, set the utility $U_i(t)$ to the expected value $\mathbf{E}[\mathbb{U}_i]$, (for all t). Let P be a policy with maximum expected utility with respect to $U_i(t)$. Then, P is a policy with maximum expected utility with respect to the probabilistic utilities $\mathbb{U}_i$.*

Proof. Let the start vertex j and start time t be arbitrary.

Let P' be some policy, let $\mathbb{U}(P')$ be its utility with respect to the probabilistic utilities, and let $U(P')$ be its utility with respect to the deterministic utilities $\mathbf{E}[\mathbb{U}_i]$. The goal is to show that $\mathbf{E}[\mathbb{U}(P')] = \mathbf{E}[U(P')]$. Let $\textsc{Stop}(P')$ be the random variable that denotes the vertex where the traveler stops. By conditional expectation, we have that

$$\begin{aligned}\mathbf{E}[\mathbb{U}(P')] &= \sum_{i\in V} \mathbf{E}[\mathbb{U}(P') \mid \textsc{Stop}(P') = i] \cdot \mathbf{P}[\textsc{Stop}(P') = i] \\ &= \sum_{i\in V} \mathbf{E}[\mathbb{U}_i \mid \textsc{Stop}(P') = i] \cdot \mathbf{P}[\textsc{Stop}(P') = i] \\ &= \sum_{i\in V} \mathbf{E}[\mathbb{U}_i] \cdot \mathbf{P}[\textsc{Stop}(P') = i] \\ &= \sum_{i\in V} \mathbf{E}[U(P') \mid \textsc{Stop}(P') = i] \cdot \mathbf{P}[\textsc{Stop}(P') = i] \\ &= \mathbf{E}[U(P')] \quad .\end{aligned}$$

where we can leave out conditioning on $\textsc{Stop}(P')$ because the utilities $\mathbb{U}_i$ do not depend on the edge failures (which are the only thing that affects where the traveler stops). We can see that a policy P' that maximizes the expected utility $\mathbf{E}[U(P')]$ with respect to the deterministic utilities also maximizes the expected utility $\mathbf{E}[\mathbb{U}(P')]$ with respect to the probabilistic utilities. ◀

B Last-Edge Markovian Failures

To compute an optimal policy for the case of Last-Edge-Markovian edge failures, we condition the expected utility equations on the *last edge that the traveler planned to take*. Since we are in a temporal graph, this implicitly also encodes the time at which the last edge was taken. Notice that a decision only needs to happen at the times when there is an incident edge.

Note that the traveler is always aware of the last edge they planned to take when the next decision needs to be taken. It would not yield any benefits for the traveler to condition on an event the traveler cannot observe (as they could not gather the necessary information to decide which case to use). If more global information was available, one could also condition on the complete past at the cost of an explosion in runtime (the state space grows exponentially with the number of past edges considered).

Proof (of Theorem 2). Let us describe the new dynamic program to compute an optimal robust routing policy in a faulty network with Last-Edge-Markovian edge failures. The approach is very similar to before, except that now we have different probabilities and we cannot apply the optimization that reduced the computation time to $O(1)$ per utility value.

The starting vertex receives a special dummy loop edge (at the starting time 0) so that all equations have the same form. For each vertex i and each incident edge $\tilde{e} = (k, i)$ or (i, k), we define the expected utility $u_i(\tilde{e})$ as the largest possible expected utility that can be obtained starting from vertex i, given that $\tilde{e}$ is the last edge which the traveler planned to take. Furthermore, we denote by $p_{e|\tilde{e}}$ the probability that e fails conditioned on $\tilde{e}$.

The base cases are as follows. The expected utility $u_i(\tilde{e})$ for the case where i is a destination vertex i is initialized as $U_i(T(\tilde{e}))$. In any case, the expected utility $u_i(\tilde{e})$ is U^0 for every vertex i where there is no edge e that leaves after $\tilde{e}$ arrives.

For all other cases, the expected utility equation is (by conditional expectation):

$$u_i(\tilde{e}) = \max_{\substack{e=(i,j)\in E \\ T(e)>T(\tilde{e})}} \Big((1 - p_{e|\tilde{e}}) \cdot u_j\big(T(e)\big) \;+\; p_{e|\tilde{e}} \cdot u_i\big(T(e)\big) \Big) \; .$$

We evaluate the dynamic program in decreasing order of the edge availability times. This works because the expected utility of an edge $\tilde{e}$ only depends on the utilities of edges e that have strictly larger availability times, i.e., $T(e) > T(\tilde{e})$.

Each vertex i has $\deg(i)$ entries that need to be computed. Each such entry depends on $O(\deg(i))$ other values. Hence, the runtime is $O(m \log m + \sum_{i \in V} \deg^2(i))$. Note that $O(\sum_{i \in V} \deg^2(i)) = O(m \max_i \deg(i)) = O(m^2)$. ◀

C Experimental Setup

C.1 Data

We use the publicly available data set "Fahrzeiten 2018 der VBZ im SOLL-IST-Vergleich"[1] from the VBZ (which is available via Open Data Zurich). It includes the *actual* (i.e. measured) departure and arrival times for all buses and trams in the Zurich transit network. The data also includes the scheduled times for those events. All times are reported in seconds (although the measured accuracy may vary and is not specifically documented). Initial testing revealed that a very small number (around $3 - 8$ per day) of departure/arrival pairs are erroneous such that the departure time is larger than the arrival time. We ignore these clearly incorrect data points in our study.

C.2 Algorithms

All evaluated approaches follow the basic idea of modeling the problem as a temporal graph and computing a policy (in the sense of Section 2) that maximizes the utility given some edge failure probabilities. For training our *probabilistic policy*, we use the delay data over the last two weeks before the first evaluated week.

We compare our algorithm to the *deterministic policy* that follows the schedule to find a journey with the *earliest arrival time*. This is equivalent to computing a policy using our algorithm by setting the failure probabilities based on the schedule times and using a utility that is zero minus the arrival time.

Moreover, we compare our algorithm to the *oracle policy* which has perfect knowledge of which connections break and which do not. This means that the failure probabilities are set based on the actual arrival and departure times. Note that the oracle policy will never miss a connection and always arrives on time if that is possible at all.

C.3 Evaluation

We evaluated the performance of the candidate policies for 38 destinations (and for each destination for every possible departure station). The stations are spread throughout the city and are of varying size. Some are exclusively tram or bus stations, others run both. The sample includes very centrally located stations and also more remote stations that often are terminal stations.

[1] `https://data.stadt-zuerich.ch/dataset/vbz_fahrzeiten_ogd_2018`, on 11.04.2019

For each destination station, we evaluated the policies during three 1-month evaluation periods in the year 2018 (namely in February, May, and November). All three periods have the same schedule. During this initial evaluation we focussed on 30 and 40 minute time budgets. As the experiments did not show any qualitative differences between the three time-periods, our final reporting focuses on the period in May (but reports on a larger variety of time budgets, where we found larger differences).

During the implementation and pre-evaluation, we used only 18 of the evaluated destinations and older data from the years 2015 and 2016. This helped us to avoid over-fitting our implementation to the evaluation periods and chosen destination stations.

In each evaluation period, we considered the average performance of the policies over the *weekdays*. For each configuration (which is given by an evaluation day, arrival deadline, time budget, and destination station), we simulate travel using the candidate policies. In the simulation, the feasibility of every transfer is determined based on the actual travel data for that day. The time for a transfer is assumed to be fixed and known. When a transfer fails, the policy is queried for a connection with time larger than the schedule time of the missed connection.

Finally, we measured the runtime on the Euler compute cluster using nodes equipped with Intel Xeon E5-2680 v3 processors (a 2.5 Ghz, 12-core processor with 30 MB last-level cache). Our computations required at most 4 GB of RAM.