Delay-Robustness of Transfer Patterns in Public Transportation Route Planning

Hannah Bast, Jonas Sternisko, Sabine Storandt

Delay-Robustness of Transfer Patterns in Public Transportation Route Planning *

Hannah Bast, Jonas Sternisko, and Sabine Storandt

Albert-Ludwigs-Universität Freiburg
Freiburg, Germany
{bast,sternis,storandt}@informatik.uni-freiburg.de

Abstract

Transfer pattern routing is a state-of-the-art speed-up technique for finding optimal paths which minimize multiple cost criteria in public transportation networks. It precomputes sequences of transfer stations along optimal paths. At query time, the optimal paths are searched among the stored transfer patterns, which allows for very fast response times even on very large networks. On the other hand, even a minor change to the timetables may affect many optimal paths, so that, in principle, a new computation of all optimal transfer patterns becomes necessary. In this paper, we examine the robustness of transfer pattern routing towards delay, which is the most common source of such updates. The intuition is that the deviating paths caused by typical updates are already covered by original transfer patterns. We perform experiments which show that the transfer patterns are remarkably robust even to large and many delays, which underlines the applicability and reliability of transfer pattern routing in realistic routing applications.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Route planning, public transportation, transfer patterns, delay, robustness

Digital Object Identifier 10.4230/OASIcs.ATMOS.2013.42

1 Introduction

When traveling with public transportation, not only the absolute time of travel matters: Also the number of transfers and the total fare are important or, for instance, the reliability of connections along the journey. The goal of public transportation route planning is to find paths that minimize a multi-criteria cost function. Public transportation data is typically available as a set of timetables and can be modeled as a directed graph. Classical route planning algorithms perform a multi-criteria variant of Dijkstra's algorithm on the graph. To our knowledge, *transfer pattern routing* [1] is the fastest speed-up technique for this problem. After precomputing sequences of transfers along all optimal paths which uses quadratic time in the number of stations, it allows to find the Pareto-optimal paths in huge networks within a few milliseconds. Because of its excellent scalability, the idea of transfer pattern routing is employed by Google Maps. If the underlying network (read: the information of the timetables) changes, the precomputed transfer patterns become outdated and optimal results cannot be guaranteed anymore. Incorporating an update into the transfer patterns is hard, because the dependency between a changed connection and the affected transfer patterns is unclear. In principle, the whole expensive precomputation has to be done again. But this is impossible as in realistic settings there are often updates. Our idea is, that provided there

* Partially supported by a Google Focused Research Award.

13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13).
Editors: Daniele Frigioni, Sebastian Stiller; pp. 42–54

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are only minor changes to the network, the original transfer patterns are sufficient to find optimal routes in most cases.

Criticism of transfer pattern routing often refers to its theoretical suboptimality and lacking support for dynamic scenarios. At the time of writing, there are no publications about if and how real-time updates can be handled by transfer pattern routing. But this is an important aspect for the practicability of the algorithm, because route planning service providers wish to recompute the transfer patterns only occasionally, when long-term changes to the timetables are made. The main contribution of this work resides in empirically proving the reliability of transfer pattern routing for location-to-location queries in the context of real-time updates. We evaluate the quality of transfer patterns in different global delay scenarios and study the immediate effect of delaying connections involved in optimal paths. Moreover, we investigate on which parameters the robustness depends and how it can be increased.

2 Related Work

Transfer pattern routing has been introduced by Bast et al. [1]. The authors outline the key components of the algorithm and present a set of techniques and heuristics to render the computation of transfer patterns feasible. Most notably is the concept of computing only parts of transfer patterns up to important stations and combining these parts at query time. Geisberger [6] elaborates how to compute transfer patterns in fully realistic settings with walking between stations and for answering location-to-location queries.

The requirements for a route planning algorithm in a dynamic network are analyzed in [10]. There have not been any publications on transfer patterns in such a setting yet. Speed-up techniques without a time-consuming precomputation and thus suitable for dynamic scenarios are for example SUBITO [3] and RAPTOR [4]. These approaches allow to find Pareto-optimal paths between stations in short time (SUBITO about 100 ms on German railway network, RAPTOR about 100 ms for London transit). However, the query times of these approaches cannot compete with transfer pattern routing on large networks (43 ms for North America).

A related field of research focuses on robustness to delay in the sense that the probability of missing a connection along a route is minimized. Most recently, a framework of algorithms based on a technique called *Connection Scan* has been introduced [5]. The authors report convincing average query times for finding the route with earliest arrival time (1.8 ms) and for multi-criteria profile queries (255 ms) on the London data set. However, it remains unclear how fast the algorithm answers one-to-one queries when more than one cost-criterion is minimized. Besides the solution of classical route planning problems, the authors apply Connection Scan to find alternative routes by minimizing the *expected* arrival time. Goerigk et al. [7] compute routes which are robust to delays in the sense that all transfers along a route are guaranteed, given a specific delay scenario. They observe that *strictly robust routes* last longer than the fastest routes, whereas *light robust routes* have a relatively small overhead. Keyhani et al. [9] introduce a stochastic model which rates the reliability of transfers along routes. Other than in those articles, robustness does not refer to routes with reliable transfers in this paper, but to *sustaining optimality*. Section 4.1 refers to the delay models of the aforementioned works in more detail.

3 Preliminaries

This section defines preliminary concepts and models. It explains the idea and components of transfer pattern routing.

3.1 Modeling timetables

A transit network is described by a set of timetables. It comprises information about stations S (e.g. train stations or bus stops) and trips of transport vehicles. A *trip* T serves a sequence of stations $stops(T) = (s_1, s_2, \ldots, s_n)$, $s_i \in S$ at arrival and departure times $(t_1^{arr}, t_1^{dep}), (t_2^{arr}, t_2^{dep}), \ldots (t_n^{arr}, t_n^{dep})$. Let $stop(T, s)$ denote the index of s in the station sequence of T. We say a trip connects two stations s_a and s_b, if $s_a, s_b \in stops(T)$ and $stop(T, s_a) < stop(T, s_b)$, and call s_a and s_b the start- and endpoint of the connection, respectively. Multiple trips which share the same sequence of stations and do not overtake each other form a *line*.

A *route* between two stations is a sequence of alternating rides on board of a vehicle and transfers between connections. The start- and endpoints of the n connections along a route form a sequence of $2n$ stations, which is called *transfer pattern* of the route. For queries between two *locations* (not stations) X and Y, there is an additional walking part at the beginning and the end. A routing algorithm answers a query with a set of routes that minimize multiple cost criteria. The *costs* of a route are the sum of the costs of all its connections and transfers. When comparing cost tuples, we say that a dominates b ($a < b$), if it is as good as b in every component and better in at least one. a and b are incomparable, if neither $a < b$ nor $b < a$. The costs of optimal paths to a query are pairwise incomparable, they form a Pareto-set.

Time-expanded Graph We use publicly available timetable data following the General Transit Feed Specification (GTFS) format and model it as time-expanded graph according to Pyrga et al. [11]. Each departure and arrival event along a trip is explicitly modeled as a node with a timestamp. The successive nodes are connected with arcs of costs corresponding to the time difference between the two events. Beside *arrival* and *departure nodes*, there are nodes modeling transfers and waiting at a station. For each departure node, there is a *transfer node* with the same timestamp and an arc connecting it to the departure node. At each station, every transfer node is connected to the next transfer node in time. To model transfers between vehicles, an arc connects each arrival node to a subsequent transfer node. Usually, a traveler cannot instantly change from one vehicle to another. We model this with the difference between the arrival and the connected transfer node not being less than a fixed *transfer buffer* of 120 seconds. We want to find routes which minimize the time of travel and the number of transfers. Therefore, the arcs have a tuple weight consisting of the time difference between the connected nodes and the *penalty*, which is 1 for arcs from arrival to transfer nodes and 0 for all other arcs. In order to decrease the size of the graph, we remove departure nodes by redirecting incoming arcs to the respective successors.

Walking Between Stations In a model for realistic route planning, transfers involving walking between two stations must be possible. Therefore, we maintain an additional walking graph with arcs between neighboring stations and the duration of walking as costs. For the sake of simplicity, we take the straight-line distance between the connected stations and assume a fixed speed of 5 km/h to compute the costs. During search, when expanding a label at an arrival node at station S and time t_{arr}, the walking graph is used to determine

the first transfer node after $t_{arr} + walk(S, T) + transfer\ buffer$ for every neighbor station T. The time difference between the two nodes and a penalty of 1 are used as weight for the relaxed virtual arc. By restricting walking to happen between arrival and transfer nodes, this model implicitly forbids *via-walking* over multiple stations in a row.

3.2 Routing with Transfer Patterns

The *transfer pattern* of a route is the sequence of stations where a change of the transportation vehicle occurs, including the departure and arrival station. When considering all possible departure times at a station A, the optimal paths for journeys $A \to B$ form a set of transfer patterns. In public transportation, this set has typically only a few elements. For example, when traveling from Paris to Nice there is a direct TGV which leaves every other hour. In between its departure times, the journey with the earliest arrival time at Nice is one of two connections with transfers at Lyon or Marseilles. We say that *Paris – Nice*, *Paris – Lyon – Nice* and *Paris – Marseilles – Nice* are the optimal transfer patterns for this station pair.

The key idea of the algorithm is that the set of optimal transfer patterns between two stations A and B *at all times* form a search space, which is orders of magnitude smaller than the original graph. Given they are known, the Pareto-optimal paths at a specific time can be found among them. In short, the algorithm determines the optimal transfer patterns for all pairs of stations and searches on the graph described by the patterns.

Computation of Optimal Transfer Patterns Conceptually, the optimal transfer patterns for a station pair A, B can be determined by running a multi-criteria variant of Dijkstra's algorithm from A. On the time-expanded graph[1], we compute the transfer patterns as described in [1]: From every station, a profile query determines the optimal paths to all reachable destinations. For every destination and its arrival nodes at times $t_1 < t_2 < \dots$ the *arrival-chain* algorithm selects a dominating subset among the set of labels consisting of (i) labels settled at t_i, and (2) labels settled at t_{i-1} with duration increased by $t_i - t_{i-1}$. Every selected label corresponds to an optimal path, which is backtracked to its origin while recording stations where transfers happened. The resulting optimal transfer patterns are stored as a *directed acyclic graph (DAG)*. In extension to [1], we exploit the fact that the departure and destination station of a transfer pattern is always known from context [12]. This allows to store all patterns in one *joint DAG*, which automatically resolves redundancies and reduces the size of the data.

The precomputation has quadratic time effort in the number of stations. When computing transfer patterns for location-to-location queries (which we do), the arrival-chain algorithm has to consider all arrival events in the walking neighborhood $\mathcal{N}(s)$ of each destination s. With an unbound neighborhood radius the running time would become cubic. Therefore we limit walking to stations within 1 000 meters. Nevertheless, the precomputation is very expensive. This is the price for the very fast query times. In order to reduce its duration, we employ the concept of important stations (hubs) and compute only parts of transfer patterns [1]. Global (unlimited) transfer patterns are computed only from important stations. From all other stations, we compute the transfer patterns up to the first transfer at a hub and a maximum of three trips. Although this heuristic leads to a loss of optimality (the search cannot find optimal paths with more than two transfers, none of which is at a hub), in

[1] Note that our results are independent of the used graph model.

practice, only very few optimal paths are affected [1]. In Section 5 we will see its marginal effect on the optimality of the algorithm's results.

Routing with Transfer Patterns Once computed, the patterns between stations A and B describe a compact graph. A location-to-location query $X@t \rightarrow Y$ is answered by constructing a *query graph* and performing a search on it. The query graph is created from the transfer patterns between departure stations $s \in \mathcal{N}(X)$, the important stations and the destination stations $s' \in \mathcal{N}(Y)$ as demonstrated in [1, 6]. We follow the refinements of Geisberger [6] and distinguish between two nodes representing alternating arrival and departure events for each station. The arcs in this graph correspond to (walking-) transfers or direct connections between stations. During the search, the travel time along these arcs can be determined using an efficient data structure. We proceed in analogy to [1] and store trips grouped by lines like this:

line_{17}	s_{14}	s_9	s_{56}	...
trip_1	8:05	9:00 9:15	10:00 10:05	...
trip_2	8:35	9:30 9:45	10:30 10:35	...
...	...	...	...	...

For each station, we compute a list of incident lines with the respective position of the station along the line. For instance: $s_{14} : \{(\text{line}_{17}, 0), (\text{line}_{26}, 8), \ldots\}$, $s_{56} : \{(\text{line}_{12}, 6), (\text{line}_{17}, 2), \ldots\}$ and so on. To determine the next direct connection between two stations, their incidence-lists are intersected and the next trip of a line connecting both stations is determined. With the query graph consisting of only several hundred arcs and the direct connection queries taking 2-10μs each, the total search time is only a few milliseconds.

4 Delay and Robustness

This section presents our delay model, points out the problem of frequent updates for a preprocessing-based algorithm and introduces our approach to handle delay with transfer pattern routing.

4.1 Delay Scenarios

Among different sources of real-time updates to timetables (trip cancellation, redirection, auxiliary connections, ...) we focus on the most common one, which is delay of trips. The literature distinguishes between primary delay (e.g. a train is late due to engine issues) and secondary delay (other trains waiting for the former) [10]. Delay models in related projects range from simplistic independence assumptions [5] over models which allow for delay to accrue [7] to sophisticated models respecting primary delay of trains, knock-on delay to other trips and delay due to waiting for late connections [8]. In a survey of stochastic models for delay, Yuan [13] successfully anneals distributions of non-negative primary delay with exponential functions. Once a trip is delayed, the propagation over successive connections follows complex rules. Refer to Berger et al. [2] for an overview and a stochastic model for delay propagation in timetables.

For the sake of simplicity and because data about real-time updates is hardly available, we focus on primary delay, ignore knock-on effects as well as scheduled security headways between trains and model delay independently between trips. In six different scenarios (Table 1), the set of trips is partitioned into groups of common average delay $\mathbb{E}\delta$. For each group, a random subset of all trips is selected. Every selected trip is delayed with time δ drawn from

Table 1 How many trips are delayed by how much in our six delay scenarios.

Scenario	Average delay $\mathbb{E}\delta$		
	5 min	15 min	50 min
Low	25%	-	-
Medium	-	25%	-
High	-	-	25%
Mix Low	10%	3%	1%
Mix Normal	20%	10%	5%
Mix Chaos	40%	40%	20%

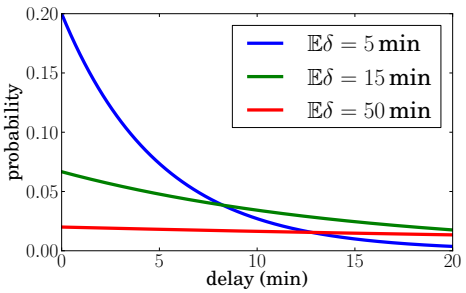


Figure 1 Probability density functions for exponential distributions with mean $\mathbb{E}\delta$.

an exponential distribution with probability density function $pdf(\delta) = 1/\mathbb{E}\delta \cdot \exp(-1/\mathbb{E}\delta \cdot \delta)$ (Figure 1). The delay is inserted starting at a uniformly random stop i, i.e. the trip's times (t_j^{arr}, t_j^{dep}) are replaced with $(t_j^{arr} + \delta, t_j^{dep} + \delta)$ for $j \geq i$. We choose three different scenarios where one quarter of the connections are delayed with 5 (Low), 15 (Medium) an 50 minutes (High) in average. In addition, we generate three combined scenarios with an increasing mixture of average delay (Mix Low, Mix Normal and Mix Chaos). In the last scenario, every trip is delayed.

One might argue that this model is too far from reality. However, by modeling delay independently between trips, the scenarios become harder to deal with than in reality. If delay occurs frequently along a specific line or in a street prone to congestion, alternative routes are more obvious and could be retrieved during the precomputation. Furthermore, in typical metropolitan networks with high service frequencies, connections typically do not wait for delayed trips. Waiting policies in hierarchical train networks are designed such that the important transfers between trips are maintained and the resulting delay for waiting trips can be compensated during the remainder of their trip and knock-on delay to further connections is minimized. Therefore, we believe that in a refined model with realistic waiting rules transfer patterns will perform more robust than in our simplified model. We are working on another set of experiments with such a model, but by the time of writing there are no results yet.

4.2 Delay and Transfer Patterns

The routing algorithm finds optimal routes only if the precomputed transfer patterns are optimal. If a trip is delayed, it is possible that an optimal route previously taking this trip will resort to another connection, thereby changing its transfer pattern. Unfortunately, not only routes along the delayed trip are affected and it is hard to decide which transfer patterns have to be updated. To make this clear, think of a train which is delayed and stops at some station at 10:15 instead of 9:45. Another train arrives at the same station at 10:00. Passengers of this train may benefit from the delayed train and arrive at their destination earlier than with a regular connection.

Thus, the optimal transfer patterns have to be computed from scratch. But this is time-consuming: for example, the transfer pattern computation for New York requires around 800 core hours ([1], Table 3). Given a steady flow of updates to the timetables, it is impossible to keep the transfer patterns up to date. On the other hand, the data structure for direct connection lookup can be computed within a few minutes ([1], Table 2). Updating a single trip is fast, as we will prove. It is thus adaptable to frequent changes of the timetables. Our

approach to deal with real-time updates is to update only the direct connection data, and search on query graphs generated from the original transfer patterns.

4.3 Updating the Direct Connection Data

Now we explain how a single trip in the direct connection data structure can be updated in real-time. When constructing the data from a collection of trips, we create a mapping from sequences of station ids to all lines that serve these stations in the given order. When updating a specific trip, the trip's stop times are changed within its line. If the line still has the FIFO-property (the trip does not overtake another trip and is not overtaken), we are done. Otherwise, the trip is removed from the line. If it does not fit into another line serving the same sequence of stations, a new line is created. The line id and the respective stop position along the line is added to the incidence list of every served station (see Section 3.2).

▶ **Lemma 1.** *Let L denote the set of lines and let $trips(l)$ denote the trips of $l \in L$. Further, $C \subseteq L$ is the set of lines which share the same station sequence as l. Then updating a trip $T \in trips(l)$ in the direct connection data structure has running time*

$$O\left(\log |L| \cdot |stops(T)| + |C| \cdot (|stops(T)| + \log |trips(l)|)\right)$$

Proof. (1) Finding the trip in the line can be done in $O(\log |trips(l)|)$, because the trips of l are sorted by time (Section 3.2). (2) Updating the trip's stop times is in $O(|stops(T)|)$. (3) Checking the FIFO-property takes $O(|stops(T)|)$. In case it is violated, candidate lines C with the same same sequence of stations have to be found in L. This takes time $O(\log |L| \cdot |stops(T)|)$ using the mapping described above. Steps (1) and (3) have to be repeated for every $c \in C$ in the worst case. ◀

For example in New York City, there are 16 454 lines and C has a maximum size of 66. For the most frequent lines, $|trips(l)|$ is 299 and for the longest trips $|stops(T)|$ is 117. Updating a trip takes 40–80 μs. Typical update rates are about 70 updates per second (German railway; primary delay, secondary delay and forecast) [10], so our approach clearly allows for real-time updates.

5 Experiments

In the previous section, we proposed to deal with delays by searching routes using the original transfer patterns (computed for the graph without delays) and updated direct connection data. This potentially leads to non-optimal responses. In this section, we present several experiments which show that this is rarely the case, even for many and large delays.

5.1 Global Delay Scenarios

Method We present experiments conducted on the data sets of Toronto (10 883 stations, 1.5 M departures) and New York City (16 765 stations, 2.3 M departures). The data can be accessed at `http://ad.informatik.uni-freiburg.de/publications`. The time-expanded graph is generated for a random weekday (from 1:00 am until 6:30 am the next day) and the transfer patterns are computed on this graph according to Section 3.2. This forms the baseline Null. We apply different delay scenarios from Section 4.1 to the data sets and compute the direct connection data and the updated time-expanded graph from it. At search time, the query graphs are constructed from the transfer patterns computed on the original graph.

Table 2 Classification of paths under different scenarios. Abbreviates ALMOST OPTIMAL σ as σ.

(a) Toronto

	OPTIMAL	A	B	BAD
NULL	99.97%	0.02%	0.01%	0.00%
LOW	99.71%	0.15%	0.04%	0.10%
MEDIUM	99.55%	0.22%	0.06%	0.17%
HIGH	99.42%	0.29%	0.08%	0.21%
MIX LOW	99.81%	0.10%	0.03%	0.06%
MIX NORMAL	99.52%	0.26%	0.06%	0.16%
MIX CHAOS	97.46%	1.40%	0.34%	0.80%

(b) New York City

	OPTIMAL	A	B	BAD
NULL	99.99%	0.01%	0.00%	0.00%
LOW	99.87%	0.01%	0.00%	0.04%
MEDIUM	99.68%	0.19%	0.03%	0.10%
HIGH	99.72%	0.17%	0.02%	0.09%
MIX LOW	99.93%	0.05%	0.00%	0.02%
MIX NORMAL	99.89%	0.07%	0.01%	0.03%
MIX CHAOS	99.19%	0.57%	0.07%	0.17%

We generate random queries $X@t \to Y$ in the following manner: The departure and destination locations X and Y are drawn from the set of locations of the stations S, taking into account the number of departing connections n_s at each station $s \in S$: The probability of selecting s is $p_s = \sqrt{n_s} / \sum_{s' \in S} \sqrt{n_{s'}}$. To avoid trivial connections, the two locations must be more than 2 000 meters away from each other. The departure time t is drawn from an interval of 24 hours starting at 4:00 am. To account for varying traffic density during the day, departure times during the rush hours are selected twice as often.

The random queries are answered by transfer pattern routing. The resulting paths are compared to reference routes. In order to compute the latter, the delayed time-expanded graph is extended with two nodes x, y representing the source and target location X and Y. For each station $s \in \mathcal{N}(X)$, x is connected to the first transfer node of s after time $t + walk(X, s)$ and every arrival node at $s' \in \mathcal{N}(Y)$ is connected to y by an arc of costs $walk(s', Y)$. On this extended graph, a multi-criteria Dijkstra is used to determine the optimal paths.

In this setting, we evaluate the robustness in each scenario. For each response to a query, the paths found by transfer pattern routing are classified independently as follows: If a path of equal costs is among the reference paths, the response is OPTIMAL. For every Dijkstra-generated path which has no correspondent of equal costs, the most similar path in terms of penalty is selected. If there is a path with the same penalty, the duration difference is inspected. If the path found by transfer pattern routing is less than 5% of the total travel time *and* less than five minutes slower than the reference path, it is ALMOST OPTIMAL A. If it is not classified as ALMOST OPTIMAL A but less than 10% *and* less than ten minutes late, it is ALMOST OPTIMAL B. Otherwise, the path is classified as BAD.

Results Tables 2a and 2b show the results of our experiments with 50 000 random queries with at least one feasible route found by the reference algorithm. For each delay scenario, the found paths were classified (about 82 000 paths in Toronto, 67 000 in New York City). The classification results show the influence of the scenarios: With increasing average delay, the share of optimal paths decreases. Although we are using the important station heuristic, almost all paths found by transfer pattern routing are optimal for the baseline NULL. The few suboptimal paths are dominated by paths with more than two transfers without an important station, which cannot be found because of the restriction of local profile queries to three trips (see Section 3.2).

Among the results for the different scenarios there are just very few suboptimal paths. Even for the worst scenario the share of suboptimal paths is below 2.6% for Toronto, and

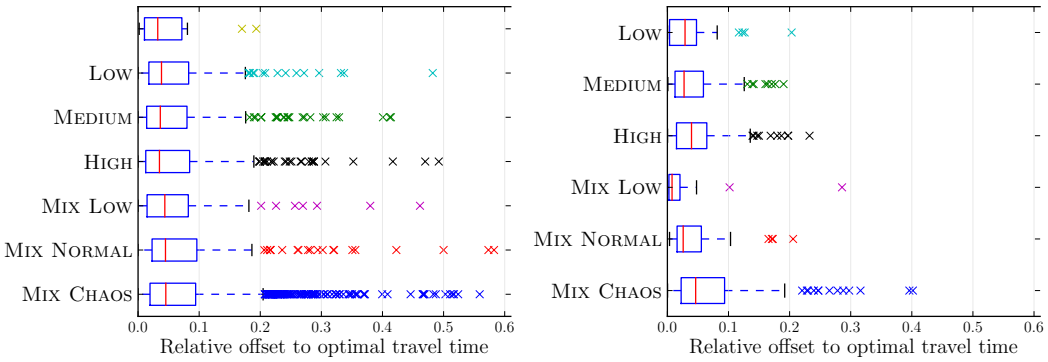
Table 3 Suboptimal paths: Relative offset to optimal travel time. Summary statistics of distributions for different delay scenarios. For example, in Toronto under the scenario Low, 25% of the suboptimal paths are at most 0.02 times slower than the optimal path.

(a) Toronto

	N	$Q_{0.25}$	$Q_{0.5}$	$Q_{0.75}$	max
Low	276	0.02	0.04	0.08	4.95
Medium	441	0.01	0.04	0.08	3.96
High	552	0.01	0.04	0.08	3.17
Mix Low	184	0.01	0.04	0.08	0.74
Mix Normal	451	0.02	0.04	0.10	0.69
Mix Chaos	2300	0.02	0.05	0.09	4.42

(b) New York City

	N	$Q_{0.25}$	$Q_{0.5}$	$Q_{0.75}$	max
Low	46	0.00	0.03	0.05	0.20
Medium	149	0.01	0.03	0.06	0.19
High	144	0.01	0.04	0.06	0.23
Mix Low	24	0.00	0.01	0.02	0.29
Mix Normal	41	0.02	0.03	0.06	0.21
Mix Chaos	332	0.02	0.05	0.09	0.40



(a) Toronto **(b)** New York City

Figure 2 Suboptimal paths: Boxplot for relative time of travel compared to the optimal path with equal number of transfers. The red line marks the median, the box contains the interval between the upper and lower quartile of the data. The whiskers have a length of 1.5 times the distance between the quartiles. The crosses mark outliers. Outliers above 0.6 are not shown.

below 1% for New York City. Beside this, most of the suboptimal paths are quite close to the optimum: The major part is classified as almost optimal a and the share of bad paths is never larger than 0.8%. For suboptimal paths, Tables 3a, 3b and Figures 2a, 2b show the distribution of the relative differences to the corresponding optimal path. The influence of the scenarios' average delay reflects similarly in the distributions as for the classification results above. We observe that the median of the distributions is below 0.05 for both data sets in all scenarios. There are a few outliers, some of which are much worse than the optimal path (at most factor 4.95 for Toronto, factor 0.37 for New York City). Manual inspection of these critical outliers showed that they typically stem from queries between remote locations with bad connectivity. For example, the worst route in Toronto misses the last connection before midnight and has to wait for six hours. Note that only paths which are dominated by a reference path of equal number of transfers are reported here. Therefore, the number of paths in the Tables 3a, 3b is slightly smaller than the number of suboptimal paths in the classification tables.

In summary, the results indicate that transfer patterns are very robust to delay. Even in the worst scenario the share of suboptimal responses is very small, and most of these paths are almost optimal. Furthermore, the results show that the limit of three trips for local transfer patterns leads only to very few suboptimal results.

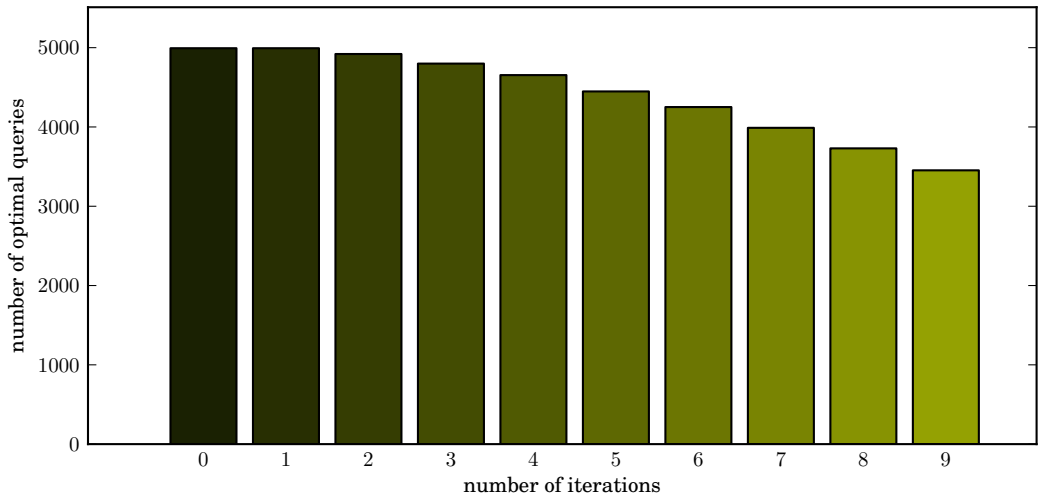


Figure 3 Number of queries which are still answered optimally after iteratively adding systematic delay of 30 minutes to the optimal paths and repeating the query. Results for 5 000 queries with a limit of nine iterations on New York City.

5.2 Controlled Delay

In our opinion, the global scenarios discussed beforehand model real transportation networks quite sufficiently. On the other hand, delaying random trips in a memoryless fashion does not clearly show why transfer patterns are robust. It is still possible that the optimal paths remain rather unchanged, for example when the delay is so small that a trip reaches the same connections as without delay. To examine the robustness of transfer patterns in more detail, we conduct another series of experiments and directly delay the optimal routes.

In a first setup, random queries are issued on the New York City data set. For every resulting optimal route one of its conducting trips is delayed with 30 minutes, such that the delay definitely affects the optimal routes. Then the query is repeated on the delayed data and the response is classified as in Section 5.1. Repeating this experiment for more than 32 000 queries showed that only 1.34% of the queries become suboptimal if we influence the connections of the optimal routes in this way.

In order to get a better understanding of the robustness, we extend this experiment to multiple rounds: A random query is drawn as before. In each round, the response of transfer pattern routing is compared to the reference response. Then, one trip of each optimal route is delayed by 30 minutes. This is repeated until the response becomes suboptimal, at most for nine times. Figure 3 summarizes the number of executed iterations until the response became suboptimal. The majority of queries was still optimal after nine successive delays. Also in this setting the transfer patterns computed on the original network proved to be very robust.

5.3 Dependencies of the Robustness

Consider an arbitrary query $X@t \to Y$. Let $r^0 = \{p_1^0, p_2^0, \ldots\}$ denote the set of paths found by transfer pattern routing in the baseline NULL. In the delayed scenario, the response is $r = \{p_1, p_2, \ldots\}$ and $r^* = \{p_1^*, p_2^*, \ldots\}$ is the (guaranteed optimal) response of the reference algorithm. The transfer patterns are robust if $r = r^*$ in terms of path costs. An *alternative*

path is a path $p_i \in r$ with costs equal to the costs of a reference path $p_i^* \in r^*$ and $p_i \neq p_i^0$ in terms of the transfer stations. How come such alternative paths are contained in the query graph? This is because in the precomputation, during the course of time paths of different transfer pattern are optimal for a station pair A, B. Besides, the query graph is a digraph with one node (-pair) for each station and can therefore contain further alternatives. For illustration, consider the digraph build from the patterns $A \to B \to C \to D$ and $A \to C \to B \to D$. In addition to the patterns it is created from, it also contains the paths *ABD* and *ACD*. This effect increases with growing number of patterns between a station pair, and thus also by building the query graph from patterns to and from important stations.

The number of alternatives depends also on the number of neighboring stations of X and Y, as the query graph is built from all patterns between these stations. The observations for both data sets in Section 5.1 differ. For Toronto, there are more suboptimal responses and they deviate more from the optimum. Here, the average number of neighbor stations $|S|^{-1} \cdot \sum_{s \in S} |\mathcal{N}(s)|$ is 50, whereas for New York City it is 92. To investigate this further, we select the ten percent of stations with the most and with the fewest neighbor stations of New York City and answer queries as in Section 5.1, but with locations X, Y drawn from one of the groups. The results clearly express a difference. Queries in the group with 152 to 306 neighbor stations are less often answered suboptimally than in the group with 1 to 36 neighbors (for Mix Chaos: 0,61% vs. 3.11%). As the maximum walking distance influences the size of the neighborhood, increasing this parameter will probably further improve the robustness.

In summary, transfer patterns allow for alternative routes. When the optimal path is iteratively delayed, at some point the optimum switches to a path with another pattern. Figure 4 shows some examples how the transfer pattern of the optimal path evolves, if the trips along the optimal path are subsequently delayed.

5.4 Improving the Robustness

The routing algorithm yields suboptimal responses whenever the optimal path is not contained in the query graph or the overlaid transfer patterns respectively. We studied reasons why the optimal paths in case of delay cannot be found in the overlaid patterns. When there is no delay, these paths are typically just slightly dominated by other paths.

The arrival-chain algorithm described in Section 3.2 selects a dominant subset among the paths between two stations. In a first approach to improve the robustness of the transfer patterns, we relaxed the domination relation for travel times in the arrival-chain algorithm: A cost-tuple a dominates another tuple b, if its travel time increased by 5%/10%/20% or at least 2/2/5 minutes is less than that of b. Other than expected, the resulting patterns are only slightly more robust, whereas even in the first setting the number of patterns has doubled. This would slow down the search time. Provided that the suboptimal responses are only a few, this minor improvement does not seem worth the additional effort.

To motivate our next approach, consider an optimal path with the transfer pattern $U \to V \to W \to X$ and imagine the trip serving $V \to W$ is delayed. We observed that some of the not-found optimal paths take redirections over some station R, but otherwise use parts of the original pattern, for example *UVRX* or *URWX*. As described in Section 5.3, overlaying transfer patterns generates a graph which contains additional paths. We tried to exploit this by enhancing the query graph with additional patterns. In the example, we would add transfer patterns for the station pairs *U,W* and *V,X* hoping that this adds subpaths *VRX* or *URW* to the query graph. While this works in theory, in practice the trigger of this extension remains unclear. Extending the query graph for every delayed arc is impractical, as this

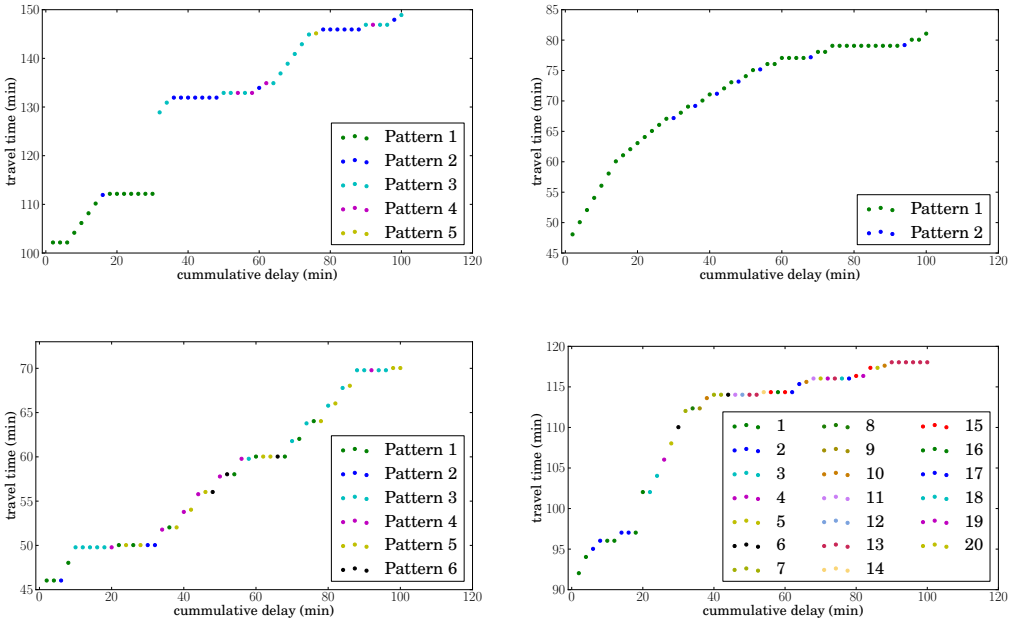


Figure 4 Evolution of the pattern of an optimal path. From the response to a fixed query, the path with the highest number of transfers is selected and one of its connections is iteratively delayed with two minutes. The plots show how the travel time increases and the pattern is occasionally changed for four exemplary queries.

would blow up its size and the construction time. Triggering the extension only for arcs with delay above a fixed threshold does not reflect the fact that occurrence and severity of suboptimal paths are only weakly related to the amount of delay. Another idea is to repeat the search on the query graph whenever the first search yields a path over a delayed connection. Alternative routes for this connection would be added to the graph. However, the suboptimal paths often do not go via delayed connections, so this is unreliable, too.

6 Conclusion & Future Work

We described how delays can be handled by transfer pattern routing without repeating its expensive precomputation. We showed how the data structure for efficient direct connection queries can be updated fast, allowing to adapt to updates in real-time. It transpired that our approach sustains the high quality of results even under extreme delay scenarios. Just a few paths are suboptimal, most of which do not deviate too much from the optimum. For example, when delaying every trip on the New York City data set with 5–50 minutes in average and answering 50 000 queries, only 450 of the resulting paths are not optimal. More than 75% of these are less than 10% and less than ten minutes off the optimum. Furthermore, we provided insight why the transfer patterns contain alternative routes and we analyzed on which factors the robustness depends.

The inherent disadvantage of the scenarios is that they model delay independently. On the one hand, in realistic public transportation delay is often systemic. For example, a traffic jam will delay a series of trips. On the other hand, there are mechanisms to compensate

delay: a bus can drive faster to catch up with its schedule. Another example are traffic agencies in the EU, which are bound by law to reimburse passengers for excessive delay. Because of this, the agencies employ decision algorithms which can make connections wait for delayed trains. As delay occurs not independently as assumed in this paper, the acquisition of realistic delay data and repetition of the experiments on top of that is a topic for future research.

Although the quality of responses are almost always optimal, there are some critically suboptimal paths. Future work should focus on eliminating these or making them less severe, for example by adding alternative transfer patterns for frequently delayed trips. We proposed three improvement approaches and discussed why they fail. If a detection mechanism for such bad responses can be found, a fall-back algorithm [3, 4, 5] could be used to find optimal responses on the updated transportation network. In order to be practicable, such a detection must not increase the running-time for the majority of queries, which are already answered optimally. This seems to be a hard problem.

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 *ESA (1)*, volume 6346 of *LNCS*, pages 290–301. Springer, 2010.

2 Annabell Berger, Andreas Gebhardt, Matthias Müller-Hannemann, and Martin Ostrowski. Stochastic Delay Prediction in Large Train Networks. In *ATMOS*, pages 100–111, 2011.

3 Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann. Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In *SEA*, volume 6049 of *LNCS*, pages 35–46. Springer, 2010.

4 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. In *ALENEX*, pages 130–140. SIAM / Omnipress, 2012.

5 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *SEA*, volume 7933 of *LNCS*, pages 43–54. Springer, 2013.

6 Robert Geisberger. *Advanced Route Planning in Transportation Networks*. PhD thesis, Karlsruhe Institute of Technology, 2011.

7 Marc Goerigk, Martin Knoth, Matthias Müller-Hannemann, Marie Schmidt, and Anita Schöbel. The Price of Robustness in Timetable Information. In *ATMOS*, pages 76–87, 2011.

8 Andrew J. Higgins and Erhan Kozan. Modeling Train Delays in Urban Networks. *Transportation Science*, 32(4):346–357, 1998.

9 Mohammad H. Keyhani, Mathias Schnee, Karsten Weihe, and Hans-Peter Zorn. Reliability and Delay Distributions of Train Connections. In *ATMOS*, pages 35–46, 2012.

10 Matthias Müller-Hannemann and Mathias Schnee. Efficient Timetable Information in the Presence of Delays. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer, 2009.

11 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12, 2007.

12 Jonas Sternisko. On Compact Representation and Robustness of Transfer Patterns in Public Transportation Routing. Master's thesis, Universität Freiburg, April 2013.

13 Jianxin Yuan. *Stochastic Modelling of Train Delays and Delay Propagation in Stations*. PhD thesis, Technische Universiteit Delft, The Netherlands, 2006.