Public Transit Routing with Unrestricted Walking*

Dorothea Wagner[1] and Tobias Zündorf[2]

1 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
dorothea.wagner@kit.edu
2 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
zuendorf@kit.edu

Abstract

We study the problem of answering profile queries in public transportation networks that allow unrestricted walking. That is, finding all Pareto-optimal journeys regarding travel time and number of transfers in a given time interval. We introduce a novel algorithm that, unlike most state-of-the-art algorithms, can compute profiles efficiently in a setting that allows arbitrary walking. Using our algorithm, we show in an extensive experimental study that allowing unrestricted walking, significantly reduces travel times, compared to settings where walking is restricted. Beyond that, we publish the transportation networks of Switzerland that we used in our study, in order to encourage further research on this topic.

1998 ACM Subject Classification G.2.2 Graph Theory

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

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.7

1 Introduction

Research on efficient route planning algorithms has seen remarkable advances in recent years, leading to speedup techniques for road networks that allow to compute optimal routes within microseconds [2]. Unfortunately, techniques that perform well on road networks often perform poor on public transit networks [5]. This lead to the development of specialized techniques like RAPTOR, CSA, and Transfer Patterns, which enable efficient route planning in public transit networks. However, these techniques are often only suitable in settings where transferring is not possible between arbitrary stops.

A common restriction in public transit routing is the requirement that the footpaths graph has to be transitively closed. One of the first techniques based on this restriction is the RAPTOR algorithm [9], which applies a dynamic programming approach to efficiently process timetable information. A transitively closed graph is also required for CSA [10], which utilizes clever memory management in order to enable journey computation within a single scan over the memory. The same holds true for the accelerated version of CSA [17]. Another technique depending on transitively closed footpaths is trip-based public transit routing [18], which is based on a graph search algorithm similar to Dijkstra's algorithm [14].

Other approaches to the public transit route planning problem do not state explicit requirements on the footpaths graph. However, the problems arising from detailed footpath graphs are often neglected. Either the used footpath graph is not specified, or the algorithms are only evaluated on rather sparse and unconnected footpath graphs. In both cases it is unknown how the techniques would perform on a public transit network in conjunction with

* This work was supported by DFG Research Grant WA 654/23-1.

a complete footpath graph. An example of a technique where no information about the size of used footpath graph is known, was presented in [15]. Another technique where the used footpath graph is not specified, is transfer patterns [1]. However, for the accelerated version of this technique, called scalable transfer patterns [3], it was specified that stops are connected by a footpath if their distance lies below 400 meters. This corresponds to a walking time of 6 minutes or less (assuming a walking speed of 4 km/h), which leads to a rather sparse footpath graph. Similarly, frequency-based search for public transit [4] was only evaluated using a limited number of footpaths. Here, two variants, one allowing up to 5 minutes walking, the other up to 15 minutes, were evaluated. Even fewer footpaths are considered if the evaluation relies on the footpath specified in the source of the public transit network. This is the case for public transit labeling (PTL) [8], SUBITO [6], or graph based techniques presented in [16]. Finally there are algorithms, like delay robust routing using MEAT, that omit footpaths altogether [12].

The utilization of a sparse footpaths graph is most often justified by arguing that walking for more then a few minutes does not improve overall travel times in practice. However, this claim has never been proven. Furthermore, it is questionable whether the decision that walking is unnecessary should be part of the problem modeling. Preferably, the model does not include artificial restrictions and an algorithm decides whether walking is reasonable or not. Another argument against unrestricted walking is, that the users of public transportation systems do not want to walk far. While this might be true for some users, it cannot be generalized to all users. Furthermore, in order to make an informed decision, it is essential that the user knows about alternative options. However, if walking is restricted, then some alternatives cannot be found. If, for example, an algorithm with a walking limit of ten minutes does not find any journey, then the user does not know if he would have to walk for an hour, or if eleven minutes of walking suffice. The only techniques that can handle unrestricted walking so far, are multimodal techniques, such as MCR [7] or UCCH [11]. These techniques can handle several modes of transportation, and restricting them to the timetable data as well as the footpath graph would solve the public transit problem. However, both MCR and UCCH can only solve queries with a fix departure time, but not profile queries.

In this work, we reevaluate the common practice of restricting the footpaths graph. To this end, we present a novel algorithm that can compute profiles for public transit networks with unrestricted walking. Using this algorithm we can efficiently evaluate the travel times between given source and target stops over the course of a whole day. Next, we prepare and compare three variants of public transit networks: The first one uses a footpaths graph that only contains transfers specified by the source of the public transit network. The second variant uses additional footpaths, which are chosen such that the transitively closed graph still has a practical size. The third variant uses an unrestricted footpaths graph. By evaluating the same set of profile queries for all variants of the network, we show that travel times are significantly improved by allowing unrestricted walking. To allow reproducibility of our results, and because recent publications do not use standardized public transit networks, we make our instances representing the network of Switzerland publicly available[1].

Our paper is organized as follows: In Section 2 we formally define public transit networks and introduce the basic notation used throughout the paper. Next, in Section 3 we present our new profile algorithm for public transit networks with unrestricted walking. We continue, in Section 4 with a detailed description of the public transit networks we will use in our evaluation, and how we combined them with unrestricted footpath graphs. Finally, we conduct an extensive experimental evaluation in Section 5, where we analyze the performance of our algorithm as well as the impact of the unrestricted footpath graph.

[1] `http://i11www.iti.kit.edu/PublicTransitData/Switzerland/`

2 Preliminaries

We define public transit networks using a format that is typical to the RAPTOR algorithm [9]. Here, a public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R})$ consists of a finite set of *stops* $\mathcal{S}$, a finite set of *trips* $\mathcal{T}$, and a finite set of *routes* $\mathcal{R}$. Each stop in $\mathcal{S}$ represents a location within the network, where passengers can enter or disembark from a vehicle (vehicles being buses, trains, etc.). Associated with a stop $u \in \mathcal{S}$ is its *minimum change time* $\tau_{\text{ch}}(u)$, which defines the minimum time required between disembarking one vehicle and entering another vehicle at the stop u. A trip in $\mathcal{T}$ is a sequence of stops which are served consecutively by a vehicle. The arrival time of the vehicle serving the trip $T \in \mathcal{T}$ at the stop u in T is denoted by $\tau_{\text{arr}}(T, u)$, the corresponding departure time is denoted by $\tau_{\text{dep}}(T, u)$. The quadruple $(T, u, \tau_{\text{arr}}(T, u), \tau_{\text{dep}}(T, u))$ is called a *stop event* of trip T at stop u. Naturally, the departure time $\tau_{\text{dep}}(T, u)$ has to be greater or equal to the arrival time $\tau_{\text{arr}}(T, u)$ at the same stop. The routes in $\mathcal{R}$ describe a partition of the trips. Two trips are part of the same route, if they serve the same stops in the same order and do not overtake one another. A trip $T_1 \in \mathcal{T}$ overtakes the trip $T_2 \in \mathcal{T}$ if two stops $u, v \in \mathcal{S}$ exist, such that T_1 arrives or departs from u before T_2 and T_1 arrives or departs from v after T_2.

The public transit network is complemented by a weighted footpath graph $G = (\mathcal{V}, \mathcal{E}, \tau_{\text{w}})$ with $\mathcal{S} \subseteq \mathcal{V}$. The footpath graph describes the time needed to walk between different locations. Each edge $e = (u, v)$ in $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents a street from u to v that can be traversed by walking. The *walking time* $\tau_{\text{w}}(e)$ specifies the time required to traverse e.

The objective of a public transit route planning algorithm is to compute journeys between a given pair of vertices. A *journey* is a sequence of stop events, sorted by time (departure time and arrival time of the stop events). Additionally, if two consecutive stop events of a journey are part of different trips, then transferring between them has to be possible with respect to minimum change time and the footpath graph. Formally, transferring from stop event $(T_1, u_1, \tau_{\text{arr}}(T_1, u_1), \tau_{\text{dep}}(T_1, u_1))$ to stop event $(T_2, u_2, \tau_{\text{arr}}(T_2, u_2), \tau_{\text{dep}}(T_2, u_2))$ is possible if $u_1 = u_2$ and $\tau_{\text{arr}}(T_1, u_1) + \tau_{\text{ch}}(u_1) < \tau_{\text{dep}}(T_2, u_2)$ or if there exists a path P in the footpath graph such that $\tau_{\text{arr}}(T_1, u_1) + \tau_{\text{w}}(P) < \tau_{\text{dep}}(T_2, u_2)$. A journey has several properties that can be used to measure the quality of the route. In this paper we consider two properties: the travel time of the journey, as well as the number of transfers. The *travel time* τ_{t} is the difference between the arrival time of the last stop event of the journey and the departure time of the first stop event. The number of transfers of a journey is the number of consecutive stop events with different trips.

An s-t-profile for the time interval $I = [\tau_{\min}, \tau_{\max}]$ represents all optimal journeys (regarding travel) from s to t that have a departure time within I. In general the profile can be represented as a piecewise linear function that maps departure time to travel time. As such, the segments of a profile can only have a slope of -1 or 0. Each optimal journey contributes one break point to the piecewise linear function (defined by the departure time and travel time of the journey). For an arbitrary departure time τ_{dep}, the value of the profile function is defined as the travel time of the earliest journey departing after τ_{dep} plus the time that has to be waited until the journey actually departs. This results in segments of the profile function with slope -1. A slope of 0 indicates a time independent part of the profile, i.e. walking from the source to the target is optimal. In order to represent all Pareto-optimal journeys (regarding travel time and number of transfers) within a certain time interval, several profiles can be used, one for every number of transfers.

The Round-bAsed Public Transit Optimized Router (RAPTOR) [9] and variations thereof [7] are algorithms that enables efficient journey computation in public transit networks.

The multimodal multicriteria RAPTOR starts with Dijkstra's algorithm on the footpath graph, in order to determine all stops that can be reached from the source by walking. Afterwards, the algorithm operates in rounds, where each round extends partial journeys by one trip. Each round consists of two phases: the first round explores the routes of the public transit network, while the second phase explores the footpath graph. In the first phase of each round all routes are scanned that contain a stop that was reached in the previous round (or by the initial Dijkstra in the first round). Afterwards, a multi source Dijkstra is used in order to find all stops that can be reached by walking from stops that were scanned during the first phase. The algorithm terminates, if during one round no new stops were reached or updated. A profile search algorithm based on this technique is called rRAPTOR. This algorithm is based on the observation that a profile cannot contain more journeys then the number of trips departing from the source (each departing trip is part of at most one journey). Thus, the algorithm simply collects all possible departure times at the source stop, followed by one execution of RAPTOR for each departure time in decreasing order. During the repeated executions of RAPTOR, labels do not need to be cleared. Since queries are performed in decreasing order regarding departure time, labels of the previous RAPTOR search can be used to prune the current search, this process is called *self pruning*. However, rRAPTOR loses its efficiency in networks that allow unrestricted walking. In such networks every stop can be reached by walking, therefore rRAPTOR would perform one RAPTOR query for every departure in the entire network.

Another efficient algorithm for public transit routing is the Connection Scan Algorithm (CSA) [10]. Using CSA requires a slightly different representation of the public transit network, which is based on *connections*. A connection represents a vehicle driving from one stop to another without intermediate stops. As such, a connection can be constructed from two consecutive stop events of the same trip. The number of connections needed to represent the network is equal to the number of stop events minus the number of trips (since a trip with n stop events contains $n-1$ consecutive pairs of stop events). The connection scan algorithm computes journeys as well as profile while performing a single scan over the sorted array of all connections. However, the algorithm requires that the footpath graph is transitively closed, and is therefore not applicable in scenarios with unrestricted walking.

3 Profile Algorithm

We now introduce our new profile algorithm for public transit networks with unrestricted walking, which is based on the multimodal multicriteria RAPTOR (MCR) [7]. As mentioned before, we cannot use rRAPTOR since every trip in the network could potentially be the first trip of an optimal journey. However, we can still use repeated executions of the basic MCR algorithm in order to compute a complete profile.

In what follows, we assume that source and target vertices $s, t \in \mathcal{V}$, as well as a time interval $I = [\tau_{\min}, \tau_{\max}]$ are given. In order to compute the s-t-profile for the interval I we start with one execution of MCR with $\tau_{\min}$ as departure time. As result of this query we obtain a journey with minimal possible arrival time τ_{arr} at t. However, we do not know the travel time of this journey, since we do not know the latest departure time from s that still allows to reach t at τ_{arr}. We determine the latest possible departure time from s by performing a backward MCR query from t, starting with the arrival time τ_{arr}. As result of these two queries we know one pair of departure time τ_{dep} and arrival time τ_{arr}, such that τ_{arr} is the earliest possible arrival time at t if departing from s at $\tau_{\min}$ and τ_{dep} is the latest possible departure time at s that allows to reach t at τ_{arr}. Therefore, the pair $(\tau_{\text{dep}}, \tau_{\text{arr}})$

is the first entry of the profile function from s to t. Furthermore, the s-t-profile is already complete for the interval $I' = [\tau_{\min}, \tau_{\text{dep}}]$. This means that we now only need to compute the profile for the interval $\tilde{I} = [\tau_{\text{dep}} + \varepsilon, \tau_{\max}]$, where the departure time $\tau_{\text{dep}} + \varepsilon$ indicates that the passenger just missed the journey that departs at τ_{dep}. The remaining profile for $\tilde{I}$ can now be computed using the same approach as for the original interval I. We repeat this process, until we are left with an empty interval. In particular, this means that the backward search results in a departure time τ_{dep} that is greater or equal to the maximum departure time $\tau_{\max}$ of the interval.

3.1 Direct Walking

The profile algorithm we described so far will perform exactly one forward and backward query for every entry of the profile. However, the approach fails if an optimal s-t-journey contains no trips at all, i.e. the optimal journey corresponds to direct walking from s to t. In this case the forward search started for a departure time of τ_{dep} will result in an arrival time of τ_{arr}. Afterwards a backward search is performed starting with the arrival time τ_{arr}. This backward search will then result with the latest possible departure time being τ_{dep}. This means the size of the interval did not decrease, except by an ε. Even worse repeating the procedure for a departure time of $\tau_{\text{dep}} + \varepsilon$ will have the same result. In order to solve this issue we use a slightly modified version of the basic query algorithm (in our case MCR). We demand that the query algorithm only returns journeys that contain at least one trip, i.e. direct walking from s to t is prohibited. This can easily be achieved by pruning the initial exploration of the footpaths graph (within MCR) if it reaches t. Apart from this, the profile algorithm remains for the most part unchanged. As before we perform alternating forward and backward searches in order to determine one profile entry at a time. However, the resulting profile might contain entries that are dominated by a pure walking journey. We remove these entries in a simple postprocessing step. For this we compute the walking time from s to t using Dijkstra's algorithm. Afterwards we remove all entries with a travel time that exceeds the walking time from the profile.

3.2 Incorporating Transfers

So far we have shown how a minimum travel time profile can be computed. However, besides travel time, the number of transfers is another important property of a journey. Thus, a profile that does not only contain all journeys with minimal travel time, but all journeys that are Pareto-optimal with respect to travel time and number of transfers is often desired. RAPTOR as well as MCR both naturally support queries that compute all Pareto-optimal journeys (regarding travel time and number of transfers) for given source vertex, target vertex, and departure time. Thus, we only have to adapt our profile algorithm so that it can take into account all Pareto-optimal journeys found by MCR. As before, when computing a profile for the interval $I = [\tau_{\min}, \tau_{\max}]$, the algorithm starts with a forward search from s for the departure time $\tau_{\min}$. The result of this forward query is a set of Pareto-optimal journeys, up to one for every possible number of transfers. Each of these journeys has a different arrival time, and eventually we will perform one backward query for each of these arrival times. We use a priority queue to organize all arrival times for which we still have to perform a backward search. As long as this queue is not empty, our algorithm extracts the minimum arrival time τ_{arr} and performs a backward search starting from the target with τ_{arr} as arrival time. As before, the result of the backward search is a departure time τ_{dep} which defines together with τ_{arr} an entry of the profile. Right after the backward search we perform

Table 1 Size figures of our instances. The footpaths graph of the 'original' and 'partial' instances are transitively closed, resulting in a high maximum vertex degree for the 'partial' instance. Note that the 'original' instances contain only very few footpath edges, even though they are transitively closed. All instances comprise a time period of two days.

PT network	Footpaths	Stops	Vertices	Edges	Connections	Max. degree
Switzerland	original	25 427	25 427	5 604	4 373 268	25
	partial	25 427	25 427	3 104 974	4 373 268	1 246
	complete	25 427	604 230	1 844 286	4 373 268	25
Germany	original	244 245	244 245	95 036	46 119 896	18
	partial	244 245	244 245	26 193 136	46 119 896	2 622
	complete	244 245	6 876 758	21 382 408	46 119 896	21

a forward search with departure time $\tau_{\text{dep}} + \varepsilon$, which possibly adds new arrival times to the queue. The advantage of this procedure is, that one backward search can potentially generate several profile entries. If several Pareto-optimal journeys differ only in their number of transfers, but have the same arrival time, then all these journeys will be found by one backward search.

In our implementation of the profile algorithm we use MCR for the forward and backward queries. However, the general approach of our algorithm can be used together with any algorithm that computes optimal *s*-*t*-journeys for a fixed departure time. The performance of our algorithm depends on the number of entries in the computed profile and the performance of the underlying query algorithm. More precisely, the underlying query algorithm will be invoked at most twice for every entry added to the profile.

4 Public Transit Networks

In order to evaluate the difference between public transit networks with and without unrestricted walking, we carefully prepared several real world networks. As basis we use the public transit networks of Germany and Switzerland. For the Germany network we use data from `bahn.de` from winter 2011/2012. This dataset was used before in [17] and comprises two successive identical days. The Switzerland instance is based on a publicly available GTFS feed[2]. Here we extracted two successive business days (30th and 31st of May 2017). Both networks contained some connections and stops beyond the borders of the country they represent. However, these stops are rather scattered and only used by very few connections, since most public transit services outside the country are not part of the dataset. In order to avoid unwanted effect from these sparse parts of the network on our experiments, we removed stops and connections outside the border of the countries. The resulting networks are listed under original footpaths in Table 1.

The original networks already contain a few footpaths. In order to obtain a comparatively dense footpath graph we complement the footpaths of our networks using data from OpenStreetMap[3]. To this end we extracted the road networks of Germany and Switzerland from the OpenStreetMap data including pedestrian zones and stairs. Since OpenStreetMap data is primarily intended for map rendering, it contains many degree one and degree two

[2] `http://gtfs.geops.ch/`
[3] `http://download.geofabrik.de/`

vertices. These vertices improve the quality of a rendered map, but they are irrelevant for routing applications, except that reduce the performance of the algorithms [13]. Therefore we replaced all degree one and two vertices with shortcuts, such that the distances in the resulting graph remain unchanged. Vertices that coincide with stops of the public transit network, were exempt from this procedure. Finally, we combine the prepared footpaths graphs with our public transit networks by identifying stops of the network with the nearest vertex of the graphs if their distance is below 5 meters. If the distance from a stop to its nearest vertex is between 5 and 100 meters, we create a new vertex positioned at the stops location and a new edge connecting the vertex to the rest of the graph. In order to compute the travel times for the edges in the footpath graph, we assume a walking speed of 4.5 km/h. Table 1 shows the sizes of the complete footpath graphs we created this way.

The complete footpaths graph cannot be used with algorithms that require a transitively closed graph, since the transitive closure would be to large. Because of this we created a third variant of the network that uses only a part of the complete footpath graph, such that the transitive closure has a reasonable size. We did this by connecting two stops with a direct edge if their distance lies below a certain threshold, and discard all other edges. Figure 1 shows the size of the resulting partial footpath graph depending on the used threshold. The figure shows that the number of edges needed for the transitive closure increases drastically with the allowed maximum walking time between stops. Therefore, approaches that require a transitively closed footpath graph are only practical if the maximum walking distance is substantially limited. For our experiments we choose an average vertex degree of about 100 as upper limit for a practical graph size, which is already much higher than average vertex degree of typical graphs used in route planning applications. This procedure, results in a partial graph for the Germany network that preserves all paths between stops that take 8 minutes or less. For the Switzerland network all paths of up to 15 minutes between neighboring stops are preserved.

Unfortunately, it is often complicated to reproduce the steps required to obtain a reasonable public transit network and footpaths graph. Reasons for this are GTFS feeds that do not comply with the specification, changing OpenStreetMap data sets, or other discrepancies preparation of the data. Because of this, almost all publications on public transit routing are evaluated on different instances. In order to counteract this trend and to enable comparability of our and future results, we make the three networks of Switzerland, that we used for our experiments publicly available[4].

5 Experiments

We implemented our algorithm in C++ compiled with GCC version 5.3.1 and optimization flag -O3. Experiments were conducted on a quad core Intel Xeon E5-1630v3 clocked at 3.7 GHz, with 128 GiB of DDR4-2133 RAM, 10 MiB of L3 cache, and 256 KiB of L2 cache. Before we continue with the performance analysis of our algorithm, we provide a detailed description of the queries we used in our experiments. Afterwards, we conduct an extensive comparisons of the profiles computed for the three variants of our networks, showing that walking should not be restricted.

[4] `http://i11www.iti.kit.edu/PublicTransitData/`

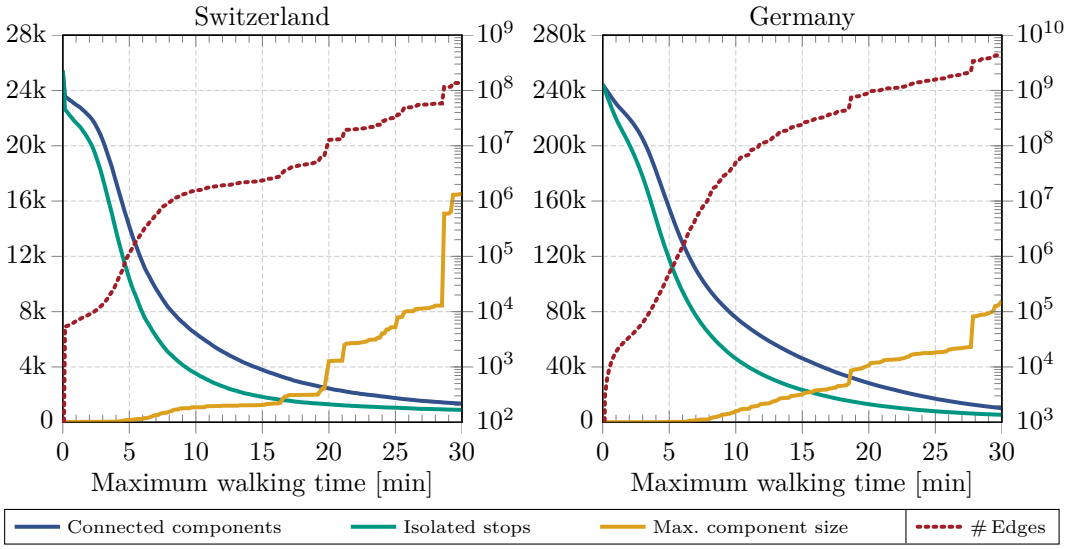

Figure 1 Size of the resulting partial network, if walking times are limited. Stops are connected by a direct edge if the distance between them in the complete graph is less or equal to the maximum walking time. Afterwards the transitive closure of these edges is computed. The plots show that the number of connected components (blue) remains high, even if the threshold for walking is rather high. Many stops are not connected to any other stops (green) and even the largest connected component remains comparatively small (yellow). However, the number of edges required for the transitive closure (red, plotted using the right y-axis) increases drastically with the allowed walking time.

5.1 Queries and Experimental Setup

We want to analyze how the results of realistic queries change with respect to the three variants of our networks. A query can of course only be evaluated for all three network variants if the source and target of the query are part of all three network variants. Thus, we only consider queries, where the source and target vertices are actual stops, as additional footpath vertices are not contained in the 'original' and 'partial' instances. Our algorithm can of course handle arbitrary source and target vertices.

Another important problem regarding the evaluation of public transit routing algorithms that we have not yet addressed, is the generation of representative queries. Commonly, algorithms are evaluated using queries where source and target stops were picked uniformly at random. However, this approach does not reflect query distributions that can be expected in real applications. It can be expected that users of a real application will predominant query journeys where the source and target stops are located within metropolitan areas. In contrast, picking source and target stops uniformly at random will often result in queries between rural locations. The choice of the queries can have significant influence on the results of the evaluation. This is because stops in rural areas are typically served by far less trips then stops in metropolitan areas. Therefore, queries are potentially simpler and can be answered faster if the source and target stop are located in rural areas. Moreover, if a stop is only infrequently served by trips, then walking is required more often. Thus, using queries that were picked uniformly at random could lead to overestimating the importance of walking.

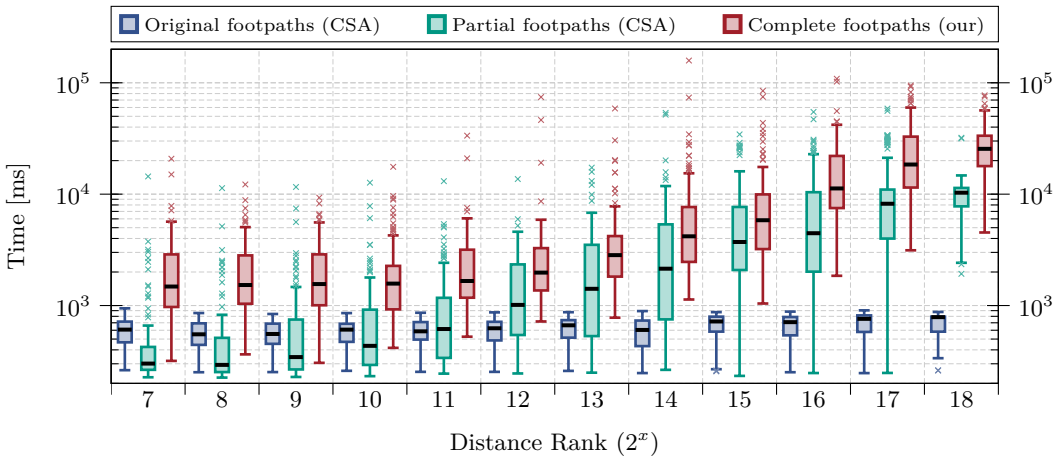

Figure 2 The running time of profile algorithms depending on the distance rank. We compare the three different variants of the *Switzerland* network. The 'original' and 'partial' instance are transitively closed, therefore a well-known technique, such as CSA can be applied. For the 'complete' instance we use our new algorithm. We evaluated 100 random queries per distance rank.

In order to avoid these problems, we do not pick the source and target stops for our test queries uniformly at random. Instead, we argue that the number of trips that serve a stop reflects the number of passengers that want to travel to or from this stop. Thus, we expect that in a real application stops with a high number of trips will occur more often as source or target stop of a query, than stops with only a few trips. We take this consideration into account during the generation of random test queries. Instead of picking source and target stops using a uniform distribution, we pick a stop u with a probability proportional to the number of trips that contain u.

Another aspect that heavily influences the result of a query is the distance from the source to the target of a query. We address this issue by partitioning the queries with respect to their distance rank. The distance rank of a query, is the number of vertices where the distance from the source to these vertices is smaller than the distance from the source to the target. Distances are measured in the complete footpath graph. In order to obtain representative queries for every distance rank, we first pick random source stops (the probability of a stop is again proportional to the number of trips containing the stop). Afterwards we pick one target for every distance rank 2^r with $r \in \mathbb{N}$. The target stop for a query with distance rank 2^r is randomly picked from all stops with a distance rank between 2^r and 2^{r-1} (as before the probability of a stop is proportional to the number of trips containing the stop).

5.2 Performance Experiments

Our first experiment is focused on the performance of profile algorithms. We compare the time required to compute complete 24 hour profiles (containing all Pareto-optimal journeys with respect to travel time and number of transfers) depending on the three variants of our networks. For this we evaluated 100 random queries for every distance rank 2^r with $r \in \mathbb{R}$. The resulting running times on the Switzerland and Germany instances are shown in Figure 2 and 3, respectively. For the network variants that contain only the original footpaths we use CSA [10]. It is clearly visible that the running time of CSA is independent of the distance rank for the 'original' instances. The reason for this is, that the instances contain only very few footpath edges, and therefore the running time is dominated by scanning the connections.

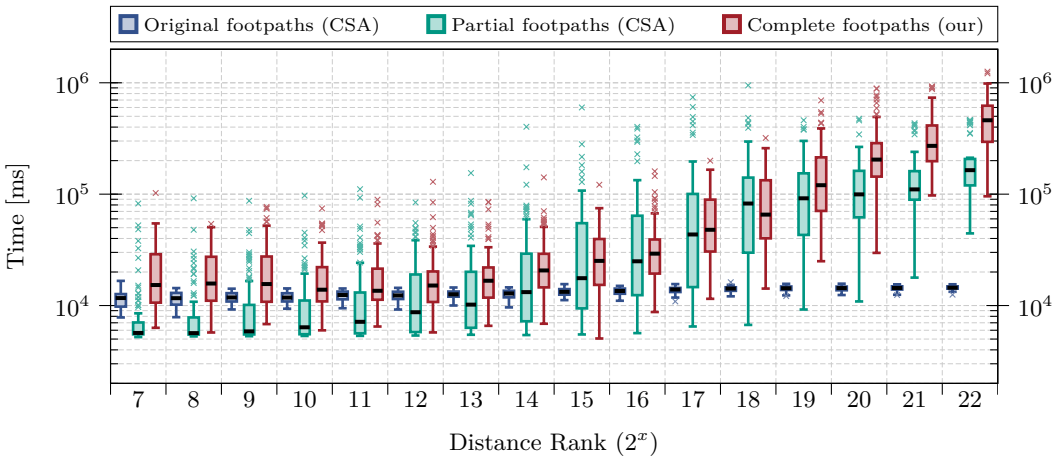

Figure 3 The running time of profile algorithms depending on the distance rank. We compare the three different variants of the *Germany* network. The 'original' and 'partial' instance are transitively closed, therefore a well-known technique, such as CSA can be applied. For the 'complete' instance we use our new algorithm. We evaluated 100 random queries per distance rank.

Since the algorithm always scans all connections, the running time is independent from the distance rank of the query.

Computing profiles for the 'partial' instance can also be done using CSA, since the footpath is transitively closed. The resulting running times, however, differ significantly from the running times of the 'original' instance. For the highest distance rank, running times are increased by an order of magnitude, resulting in query times of about 2 minutes for the Germany network. The query time decreases with decreasing distance rank, as a result of target pruning. For small distance ranks, the running time even falls below the running time for the 'original' instance. The reason for this is most probably a high number of queries where walking is the optimal solution. This decreases the complexity of the profile functions, which leads to decreased running times.

Finally we examine the running time for the 'complete' variant of our networks. Since the footpath graph of these instances is not transitively closed, we have to use our new algorithm. Computing a profile using our algorithm takes about 6 minutes on average. Despite the fact that our algorithm computes profiles for more complex networks with unrestricted walking, running times are only a factor 2 to 4 slower than CSA on the 'partial' instance. Similar to CSA, the running time of our algorithm decreases with decreasing distance rank. The reason for this is the underlying search algorithm (in our case MCR), which is faster for local queries due to target pruning.

5.3 Travel Time Comparison

Finally, we analyze how the travel time of optimal journeys changes, depending on the footpath graph. For this we compare the results of the same 100 random queries per distance rank that we used for the performance experiments. Our evaluation focuses on the minimum travel time only, i.e. we ignore Pareto-optimal solutions with fewer transfers. This leads to a conservative estimation for the importance of walking, since walking is even more indispensable if the number of transfers is limited.

We examine the average travel time of all journeys with the same distance rank. Overall, we find that differences in travel time between the three variants of our networks are consistent

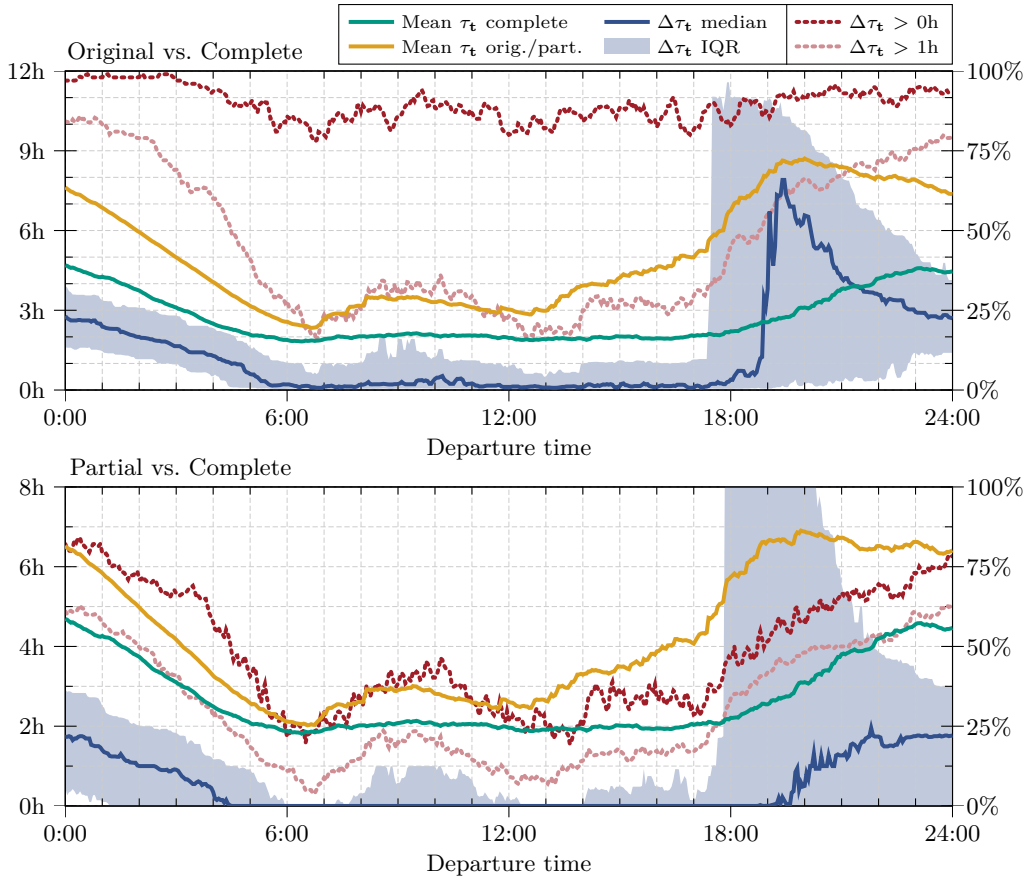

Figure 4 Comparison of the optimal travel times throughout the day for the three variants of the *Germany* network. The upper plot compares the 'original' and 'complete' instances, the lower plot compares the 'partial' and 'complete' instances. We evaluated 100 random queries with distance rank 2^{16}, which correspond to an average travel time of 2 hours. The average of the travel time in the 'complete' instance is depicted in green. The blue curve depicts the median of the travel time difference between the two compared instances, the light blue shaded area depicts the interquartile range (IQR). The dark red dotted curve (using the right y-axis) indicates the percentage of queries where the 'original' respectively 'partial' travel time is suboptimal. The light red dotted curve (using the right y-axis) indicates the percentage of queries where travel time difference is more than 1 hour.

over all distance ranks. In Figure 4 and 5 we present exemplary results for the distance rank 2^{16}. This distance rank roughly corresponds to an average travel time of two hours. In all plots, the green curve indicates the average travel time in the 'complete' network. The yellow curve specifies the average travel time in the 'original' and 'partial' network, respectively. The plots show that using only the 'original' footpath leads to travel times that surpass optimal travel times by several hours. Travel times in the 'partial' network are already closer to the optimum, at least during the day. However, in the evening and during the night, unrestricted walking still improves the travel time significantly. The median difference in travel time between the 'original' and 'partial' is up to two hours during the night for the Germany network (as shown by the blue curve).

The importance of unrestricted walking becomes even more noticeable when looking at the percentage of queries where restricted walking leads to increased travel times. The dark red dotted curve depicts the percentage of queries that have an increased travel time

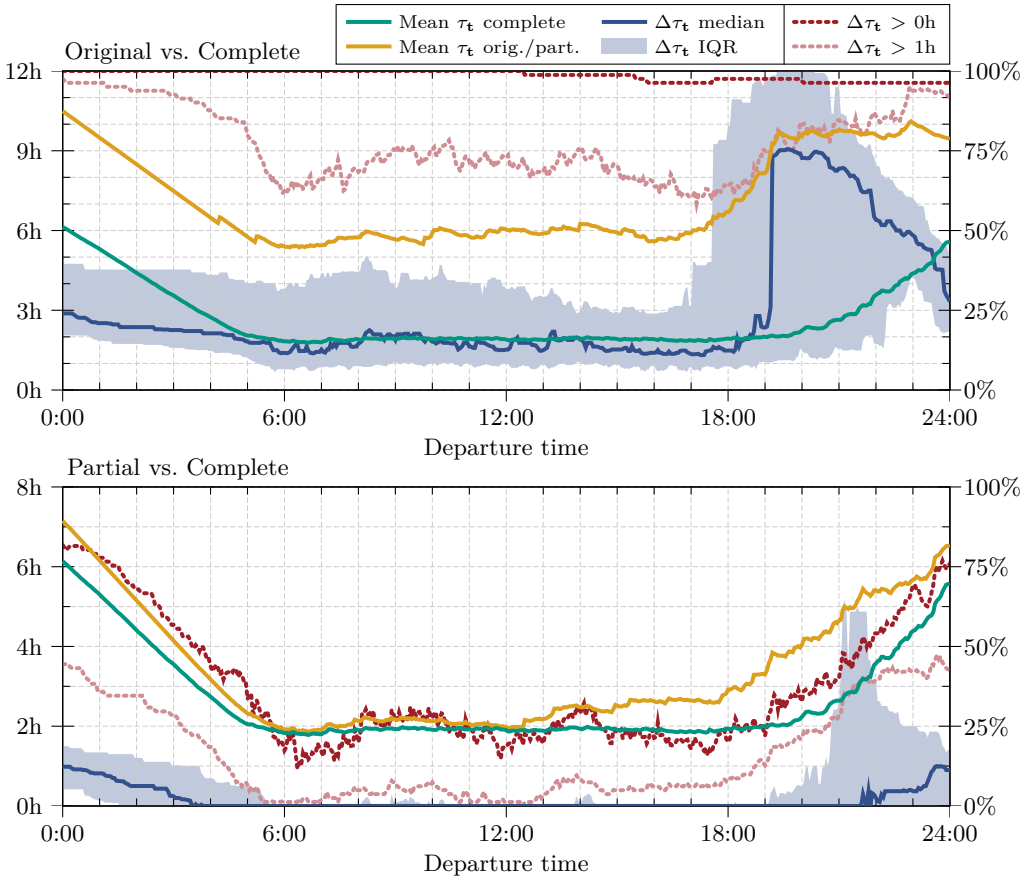

Figure 5 Comparison of the optimal travel times throughout the day for the three variants of the *Switzerland* network. The upper plot compares the 'original' and 'complete' instances, the lower plot compares the 'partial' and 'complete' instances. We evaluated 100 random queries with distance rank 2^{16}, which correspond to an average travel time of 2 hours. The average of the travel time in the 'complete' instance is depicted in green. The blue curve depicts the median of the travel time difference between the two compared instances, the light blue shaded area depicts the interquartile range (IQR). The dark red dotted curve (using the right y-axis) indicates the percentage of queries where the 'original' respectively 'partial' travel time is suboptimal. The light red dotted curve (using the right y-axis) indicates the percentage of queries where travel time difference is more than 1 hour.

compared to the 'optimal' instance. Using only the 'original' footpath leads to more then 75% of the queries having an increased travel time, even during the day. For the 'partial' instances, still about 25% of the queries have an increased travel time. Our results show that there exists even a significant percentage of queries, where the difference in travel time is more then on hour. This percentage is depicted by the light red dotted curve. For the Germany network, using 'partial' footpath leads to about one eighth of the queries having a travel time that surpasses the optimal travel time by more than one hour. Overall, our results demonstrate that unrestricted walking has a significant impact on the travel times.

6 Conclusion

In this work we had an in-depth look into public transit routing. We started with the observation that most public transit algorithms neglect problems arising from unrestricted walking. Thus, we provide a novel profile algorithm that works on arbitrary footpath graphs. Accompanying our algorithmic results we created and published a first benchmark instance combining a public transit network with an unrestricted footpath graph. Finally, we conducted an extensive experimental study. While being applicable to networks that could not be handled before, our algorithm still achieves running times comparable to a state-of-the-art technique. Furthermore, we demonstrated that walking has a significant impact on travel times. Compared to conventional models, travel times are reduced by hours when allowing unrestricted walking. Thus walking should always be considered in public transit routing.

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 *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

2 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

3 Hannah Bast, Matthias Hertel, and Sabine Storand. Scalable Transfer Patterns. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 15–29. SIAM, 2016.

4 Hannah Bast and Sabine Storandt. Frequency-Based Search for Public Transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22. ACM Press, November 2014.

5 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, OpenAccess Series in Informatics (OASIcs), 2009.

6 Annabell Berger, Martin Grimmer, and Matthias Müller–Hannemann. Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.

7 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing Multimodal Journeys in Practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2013.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, Lecture Notes in Computer Science, pages 273–285. Springer, 2015. doi:10.1007/978-3-319-20086-6_21.

9 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 2014. doi:10.1287/trsc.2014.0534.

10 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

11 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *Journal of Experimental Algorithmics (JEA)*, 19:3–2, 2015.

12 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time. In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, OpenAccess Series in Informatics (OASIcs), 2014.

13 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Fast Exact Shortest Path and Distance Queries on Road Networks with Parametrized Costs. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 66:1–66:4. ACM Press, 2015.

14 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

15 Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

16 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

17 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

18 Sascha Witt. Trip-Based Public Transit Routing. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science. Springer, 2015.