Albert-Ludwigs-Universität Freiburg

Master Thesis

Efficient Multi-modal Route Planning with Transfer Patterns

Cynthia Jiménez Cárdenas

Reviewers:

Prof. Dr. Hannah Bast
Prof. Dr. Christian Schindelhauer

Supervisors:

Prof. Dr. Hannah Bast
Dr. Sabine Storandt

Freiburg im Breisgau, October 23, 2013

DECLARATION

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place,Date	Signature

Acknowledgments

I would like to thank first my Professor Hannah Bast for providing me with such an interesting project, for her excitement in regard to teaching that inspired me to continue working on route planning and for her advice during the course of this project. My sincere thanks also goes to Professor Christian Schindelhauer for agreeing to review this thesis as second supervisor. Furthermore, I would like to express my sincere gratitude to my supervisor Sabine Storandt for her guidance, enthusiasm, and help.

In addition, I would like to thank my family, my beloved parents, Eliza and Roberto, for the opportunities they gave me and for always being there for me, and my siblings, Tatis and Alex for their unconditional support. Especially, my sister for proofreading this thesis and Alex for all his helpful advices.

Above all, I want to thank my beloved husband Daniel for his love, patience and constant support, for all the late nights and early mornings, and for keeping me sane over the past few months. Thank you for being my reviewer, proofreader, and sounding board. But most of all, thank you for being my best friend.

Abstract

In these days, there is a big interest in route planning systems with different types of transportation. We study a way to efficiently compute optimal paths with real data in a multi-modal scenario considering walking, transit and car. Therefore, we implement Transfer Patterns, which is a state-of-the-art route planning approach [8]. This allows to obtain shortest paths queries within milliseconds. When using a Pareto-cost model various optimal paths are provided for the user. However, the high number of optimal solutions also includes a lot of unreasonable routes, although they are Pareto-optimal. To address this problem, we use a filtering procedure called Types aNd Thresholds (TNT) [10], which provides a both small and diverse set of reasonable optimal paths decreasing the number of transfer patterns and hence resulting in faster query times. To our knowledge this is the first work which realizes the combination of these two approaches (Transfer Patterns with TNT). We present an evaluation on query times of selected optimal paths, which shows different and reasonable route options with an average query time of 14 milliseconds for the Freiburg network.

Zusammenfassung

Derzeit besteht ein sehr großes Interesse an der Entwicklung von neuartigen Routenplanern, die in der Lage sind mehrere Transportmodi miteinander zu kombinieren. Wir untersuchen daher ein Verfahren, das für ein reales Netzwerk optimale Wege in einem Mehrfachmodell unter Einbeziehung der Transportformen „Gehen“, „Öffentliche Verkehrsmittel“ und „PKW“ errechnet. Um dies zu erreichen, verwenden wir sogenannte Transfermuster (Transfer Patterns), die im Bereich der Routenplanung einen hochmodernen Ansatz darstellen [8]. Auf diese Weise können innerhalb von Millisekunden Suchanfragen bearbeitet und kürzeste Strecken angezeigt werden. Zudem werden bei der Verwendung des Pareto-Kostenmodells verschiedene optimale Routen für den Benutzer bereitgestellt. Allerdings beinhaltet diese hohe Anzahl von optimalen Wegen auch sehr viele unlogische und unangemessene Routen, obwohl diese das Kriterium der Pareto-Optimalität erfüllen. Um diesem Problem zu begegnen wird das Filterverfahren „Types and Threshold“ (TNT) eingesetzt [10]. Der Filter reduziert diese Anzahl, so dass nur noch wenige, jedoch unterschiedliche optimale Wege angezeigt werden. Das wiederum verringert die Anzahl an „Transfer Patterns“ und folglich auch die Suchanfragezeit. Unseres Wissens ist dies die erste Arbeit, die das Verfahren „Transfer Patterns“ mit dem Filter „Types and Threshold“ kombiniert. Wir stellen Auswertungen von Suchanfragezeiten bestimmter Strecken dar. Diese zeigen im Freiburger Netzwerk verschiedene und angemessene Streckenoptionen bei einer durchschnittlichen Abfragezeit von nur 14 Millisekunden.

Contents

Chapter 1

Introduction

A lot of travelers do not use traditional maps anymore when finding a perfect route from one place to another. Planning a trip with the help of route planning applications like Google Maps[1], are in the ascendant. When entering the starting and end point these applications provide possible routes very fast so that time consuming map reading can be avoided. The list of displayed routes is in general ranked in terms of the criterion "duration", providing the fastest. However, the best route is not always the fastest one, choosing the best route depends on users preferences. For example, traveling from Mannheim to Freiburg, there are several options. One of them is taking a direct connection by train, which takes 3 hours. Another route option takes 2 hours and 30 minutes but implies a transfer in Karlsruhe. While one person traveling with heavy luggage would prefer the route with less transfers although it implies more travel time, another person would prefer the fastest route to get on time to an appointment. One can also think on another traveling criterion like the price. Route planning applications which take into account several criteria are called multi-criteria.

When using public transportation as the traveling mode, advanced route planning systems also include the walking mode. This is due to the fact that the starting and end point are often not a station, so that at the beginning or at the end a portion of walking is necessary. One might want to have an integral solution that includes the way from your house to the train station in Mannheim as well as the way from the train station in Freiburg to the final location. Additionally, walking is also beneficial when a change from one station to another station gives an advantage in time. These requests that include also the part from location to stations and vice versa are called location-to-location queries. Including other means of transportation like car (taxi) and bike provide even more interesting routes and time saving possibilities. In certain cases, in metropolitan areas, taking a taxi or bike in the middle of a trip can save a lot of time, if this ride describes a shortcut compared to public transport or brings the user to a different station with better connections. If it is not within a reasonable walking radius, a user will not even check schedules from this station. A user can not figure out these options easily by himself and probably wants to use

[1] http://www.google.com/transit

them if he knew.

Up to this point we consider that using multi-criteria and multiple modes of transportation for location-to-location queries gives a great value to plan a trip. Moreover, several diverse optimal paths should be displayed, so that the user choose the one that fits better for him. In our implementation we consider the modes public transportation, walking and car. The criteria taken into account are travel time, number of transfers and car duration. The latter can be seen as the implied cost of using a taxi. In order to minimize the values of the aforementioned criteria and obtain multiple options we consider the use of Pareto sets [22]. However, including the car mode leads to a huge amount of optimal paths. Most of them have only a slight difference between the car usage and total travel time. One reason of the increased number of optimal paths is that each portion of transit between stations can be replaced by car usage. Within a city traveling by car is normally faster than using public transportation. So that replacing a portion of transit increases the car duration, but decreases the total travel time. Because in the Pareto sense a cost dominates another cost only if it is less or equal in all components, all these variations are optimal. Consider the example of using the car to the nearest bus station and then continue from there using a bus line. It is also optimal to go by car to the second nearest bus station and continue the transit from there. The first option has a lower car duration but more travel time, whereas the second option has a greater car duration but less travel time. Both solutions are Pareto-optimal but very similar so that one solution cannot be considered as an alternative route of the other one. Another problem in this setting is that resulting optimal paths are not reasonable. For example, consider a query from Mannheim to Freiburg at night. One possible result is "take a taxi for 2 hours and then 2 minutes with the tram and then walk" or "walk one hour then take a bus 3 minutes then a taxi 1 minute". Although these options are Pareto-optimal, it is unlikely that a user will choose them and thus they should be filtered out.

Our route planning strategy to deal with these challenges consists on combining a state-of-the art algorithm called Transfer Patterns algorithm [7] with filtering techniques called Types and Thresholds (TNT) [10]. To our knowledge this is the first time this two approaches are combined. The Transfer Patterns algorithm is a novel approach to get fast queries in public transportation networks. It exploits the idea that a given source and target exhibit a time-independently limited number of transfer patterns, which are the sequences of stations where a change of vehicle happens. For example, when traveling from Mannheim to Freiburg, the best connection is either transfering in Karlsruhe, transfering in Offenburg or a direct connection to Freiburg. In this case the transfer patterns are Mannheim-Karlsruhe-Freiburg, Mannheim-Offenburg-Freiburg and Mannheim-Freiburg. These transfers are precomputed and stored, so that at query time optimal paths are only explored within

the schedules of these three options, which is a very small search space compared with normal Dijkstra's algorithm [18]. Therefore, it enables query times of just a few milliseconds. Our approach creates a lot of optimal paths which have to be reduced in order to show the user only different and reasonable routes. Therefore, we use a method called Types and Thresholds, which filters out non-adequate results.

In the next section we give an overview of related investigations on multi-modal and multi-criteria route planning. Chapters 3 and 4 give background on how to model road and transit networks as graphs. Additionally, they provide a brief description of algorithms to compute shortest paths both in road and transit networks. Chapter 5 describes the graph model combining road and transit networks. Chapter 6 explains the implementation of multi-modal transfer patterns and the concept of Types and Thresholds. Chapter 7 shows results of the evaluation on the Vitoria-Gasteiz and Freiburg networks. Chapter 8 summarizes and concludes this work.

Chapter 2

Related Work

Recent research on route planning focuses on multi-criteria and multi-modal routing. Some of these works convert the multi-criteria search into a single criterion problem [29] [4] [34]. However, combining multiple criteria into one criterion may miss optimal paths [13]. Moreover in [34] it is also required that the user knows a priori a mode hierarchy, which means when to use a specific transportation mode. In practice, the user normally does not know this in advance, but wants this information to be provided by the system. Other approaches restrict the modes, for instance by limiting the car usage at the beginning and end of a route [15]. Further approaches, make use of Pareto sets [22] to minimize multiple criteria and provide several results to the user [14] [6] [9].

Delling et al. [14] consider a multi-criteria and multi-modal scenario based on RAPTOR (Round Based Public Transit Optimized Router) [16]. Here, full Pareto sets of optimal costs are computed taking into account the criteria travel time, number of transfers, walking duration and taxi costs. They address the problem of large Pareto sets by ranking the results by score. This score denotes the significance of the result and is computed using fuzzy logic in a post processing step. Furthermore, they applied heuristics during query search to speed up query times. They obtained average query times of one second on the London network.

In a very recent work, a filter procedure called Types and Thresholds (TNT) was introduced in [6]. This approach involves multiple criteria and the modes transit, walk and car. Here, a classification of reasonable paths was provided as a result of an analysis of availabiltiy, velocity and relative durations of each mode of transportation. Based on this classification and other assumptions, heuristics were defined to reduce the number of Pareto-optimal paths to a small and diverse result set. They achieved average query times of one second for the New York network. Additionally, they provide an evaluation that indicates that the quality of the resulting paths is preserved despite of heuristics.

Query times of [6] and [14] (around one second) allow for interactive queries in

large networks. However, faster query times on transit networks have been reported by the precomputation of Transfer Patterns (around 6 milliseconds for New York).

Routing with Transfer Patterns was introduced by Bast et al. [7]. It is based on a precomputation and fast queries to time tables for direct connections. The authors provide heuristics to speed up the precomputation, such as the important stations heuristic which makes feasible precomputation on large networks. They obtained query times of few milliseconds on large networks, such as the train network of Switzerland and the train + local transport network of a big part of North America. Geisberger research in [20] elaborated in detail the Transfer Pattern algorithm and introduced an extension to compute location-to-location queries.

Braun in his master thesis [9] extended the Transfer Patterns algorithm with the car mode. As well as in [6] and [14], he faced the problem of large Pareto sets. He identified several reasons of this problem and proposed ways to reduce this sets by restricting the graph model. He showed that precomputation using the restricted graphs is feasible also for large networks like New York. However, the work concluded that these restricted models decrease the quality of the resulting paths. Furthermore, this approach was only shaped for station-to-station queries.

Other works on routing with transfer patterns focuses on the robustness to delay. This means how time table updates due to delays affect the optimal paths. In [32] it was proven that transfer patterns are robust against delays, which means that alternative paths caused by delays are already covered by transfer patterns.

In this thesis we combine for the first time the Transfer Patterns algorithm which allows for fast query times with concepts of Types and Thresholds during preprocessing time to obtain a representative result set involving the modes transit, walk and car and provide an efficient backend to compute optimal paths for the Vitoria and Freiburg network.

Chapter 3

Route Planning in Road Networks

This chapter presents an overview of how to model a road network as a graph. Additionally, it provides background about the algorithms to compute shortest paths. First, we describe the most common algorithm to solve the problem of finding shortest paths in a graph, this is called Dijkstra's Algorithm. Then, we provide a description of other variants of Dijkstra's Algorithm that are more goal-directed due to the use of heuristics. Finally, we introduce the concept of Contraction Hierarchies, which is a speed-up technique based on precomputation[21].

3.1 OSM Data

OpenStreetMap(OSM) is a collaborative initiative, whose purpose is to provide free geographical data [3]. The data is collected by contributors who record their movements using a GPS tracking device. These maps can be downloaded in XML format under the Open Database License. OSM files contain not only road data, but also other map related data such as rivers, buildings, etc. We extract from these files only node tags and way tags with the attribute "highway". Node tags represent a specific location (latitude, longitude) and way tags connect nodes in a road.

3.2 Modelling Road Networks as Graphs

A graph $G = (V, E)$ contains a set of nodes V and a set of edges E. For road networks a node $v \in V$ represents an intersection of two road segments and corresponds to a location with a given latitude and longitude. An edge $e \in E$ connects two nodes and describes the road between two intersections. For an edge e we also write (u, v), where $u \in U$ is called the *tail node* and $v \in V$ is called the *head node*. Each edge e, also referred as arc, is weighted with a cost $c(u, v)$. For road networks this cost normally represents the travel time required between these two intersections on a map.
A *shortest path* between two nodes v and v' is a sequence of nodes $v_1, v_2, ..., v_n$

(where $v = v_1$ and $v' = v_n$) that are connected by edges $(v_i, v_i + 1)$. Furthermore, the sum of the edge's cost is the minimum over all possible paths between v and v'.

3.3 Algorithms

3.3.1 Dijkstra and Extensions

Dijkstra's Algorithm [18] developed in the 50's is a common algorithm to compute shortest paths in road networks. The algorithm maintains a priority queue with tentative costs. Initially, the source node has tentative cost zero and all the other nodes have a cost of infinity. Then, the node with the smallest tentative cost is taken out of the priority queue and its status is changed to *settled*. For each outgoing arc the algorithm checks whether the tentative cost of the tail node can be improved. If this is the case, the node is added to the priority queue with its respective tentative cost. This step is called *relaxing* an arc. The algorithm stops when the target node or all nodes in the graph are settled. As a result, the tentative cost of each settled node is the lowest cost from the source to this node. This only works if all arcs are non-negative.

The drawback of this algorithm is the omnidirectional search for the shortest path, which requires vast amount of computational resources and hence a high query time. However, it is still the base of many other algorithms that compute shortest paths. Such algorithms extend the original Dijkstra to reduce the number of visited nodes and the search space of original Dijkstra's algorithm, respectively. As a consequence the time to find the shortest path decreases in comparison to the original algorithm.

A* Search One extension of Dijkstra's algorithm is A* [23]. It makes use of an heuristic function that estimates the distance to the target. The simplest approach of an heuristic function is to compute the straight-line distance between source and target. This estimated distance is added to the tentative distance of a node before the insertion in the priority queue. As a result, nodes that are closer to the target will come out sooner from the priority queue than nodes with greater distances from the target. Hence, the search of the shortest path is more goal-directed than with the original Dijkstra's algorithm.
Another more sophisticated heuristic computes a set of landmarks. It exploits the triangle inequality to approximate more accurately the distance to the target. This approximation is used in the same way as the straight line distance with A*. This combination of A* and landmarks heuristic is called ALT algorithm [1].

Bidirectional Dijkstra Another extension of Dijkstra's algorithm is its bidirectional version [12] which simultaneously searches from both source and target until

they meet. That is, when settling a node from one Dijkstra computation that is already settled in the other Dijkstra. The Dijkstra from the target is performed on the graph with inverted arcs, the so called *backwards graph*. This version of Dijkstra's algorithm decreases the search space of normal Dijkstra and is the base of other elaborated algorithms like Contraction Hierarchies.

3.3.2 Contraction Hierarchies

Unlike Dijkstra's Algorithm and its variants, Contraction Hierarchies [21] introduced by Geisberger et al. uses preprocessed data to improve query times. The central routine of the algorithm consists on *contracting* nodes. Thereby a node and its adjacent arcs are taken out from the graph. Furthermore, additional arcs, called *shortcuts*, are added whenever a shortest path goes through the contracted node. Thus, the lengths of the shortest paths are preserved between the remaining nodes in the graph.

All nodes in the graph are contracted following a specific node order. This order plays an important role on the number of shortcuts to be added. To compute the node order, there are several heuristics such as the *edge difference*. This is the number of shortcuts that would have to be added if the node was contracted minus the number of arcs incident to that node.

After the contraction process, a new graph is created consisting of nodes and edges of the original graph and all shortcuts. Shortest paths can be found by a bidirectional search. On the one hand in the forward search (Dijkstra from the source) only arcs with non-decreasing order are considered, which is called the upwards graph. On the other hand in the backward search (Dijkstra from the target) only arcs with non-decreasing order are considered and is called the downwards graph. Therefore, the total shortest path consists of the shortest paths on the upwards graph and on the downwards graph.

Chapter 4

Route Planning in Public Transportation Networks

This chapter begins with an overview of a common format for time table data. Later on, it elaborates on how to model public transportation networks as graphs. Finally, it gives a brief overview of a variant of Dijkstras algorithm to consider multiple criteria (multi-label Dijkstra) and transfer patterns. For more details on transfer patterns see Chapter 6.

4.1 General Transit Feed Specification (GTFS)

Time table data is represented in several files called GTFS-feed. Each of these files represent a particular feature of the transit network, such as trips, stops, routes, times, fares. The following files are required to form a GTFS-feed:

- *agency.txt*: Data from entity that provides the GTFS-feed.
- *stops.txt*: Data about locations for alighting and boarding vehicles. For example, latitude, longitude, name of the stop, and so on. We refer to a stop also as station.
- *trips.txt*: Data from trips, which are series composed of two or more stops at specific time. Example of this data is the name of the trip and head sign. The latter, refers to the destination of the trip, for instance *direction* : *main train station*.
- *routes.txt*: Data about routes, which are groups of trips that users identify normally as lines. Examples of this data are the name of the route and the type of vehicle (tram, subway, bus).
- *stop_times.txt*: Data of the arrival and departure time of a vehicle at a particular stop. The associated trip of the vehicle and the stop sequence are also included.

- *calendar.txt*: Data of the days of the week where a trip is available, as well as the start and end date.

4.2 Modelling Public Transportation Networks as Graphs

Data obtained by a GTFS-feed such as stations (trains stations, bus stops), lines (trams, buses, trains) and time tables can be used to build a graph. There are two common approaches to model these data as graphs: The time-expanded and the time-dependent approach [28]. Both models require a special treatment to manage transfers. The time-expanded approach yields to a very large graph, but it is more robust for complex scenarios, whereas the time-dependent has better performance in terms of computing shortest paths. A detailed discussion about advantages and disadvantages of both models is given in [30].

4.2.1 Time-Expanded Model

A node in the time-expanded graph corresponds to a specific time event, such as departure or arrival at a particular station. These nodes are called departure nodes and arrival nodes. Arcs between nodes represent either traveling on a vehicle between stations or waiting at a station. The cost of arcs between stations is commonly the time it takes to go from one station to the other with the specified vehicle. To enable transferring to another vehicle in a station, for each departure node a transfer node with the same time is added. This transfer node is connected by an arc to its respective departure node, this arc represents boarding a vehicle. Arrival nodes are connected to the first transfer node with greater time, these arcs represent alighting from a vehicle at a station. Transfer nodes are ordered by time and connected by arcs that represent waiting time. Because this is a graph with non-negative arc costs, Dijkstras algorithm can be used to compute shortest paths. An example of this graph is shown in Fig 4.1.

4.2.2 Time-Dependent Model

In this model each node in the graph represents a station. If there is a vehicle traveling between two stations an arc is added. For this graph there are no fixed costs but a cost function that depends on the time a particular arc is evaluated. Fig. 4.2 shows an example of a time-dependent graph. This graph is significantly smaller than the time-expanded but needs a special treatment to enable transfers. One extension of this model, which handles transfers is the *train-route-model*. Here, for each station there is a station node and for each line that stops at this station there is a route node. Route nodes have arcs to and from the station node. Outgoing arcs

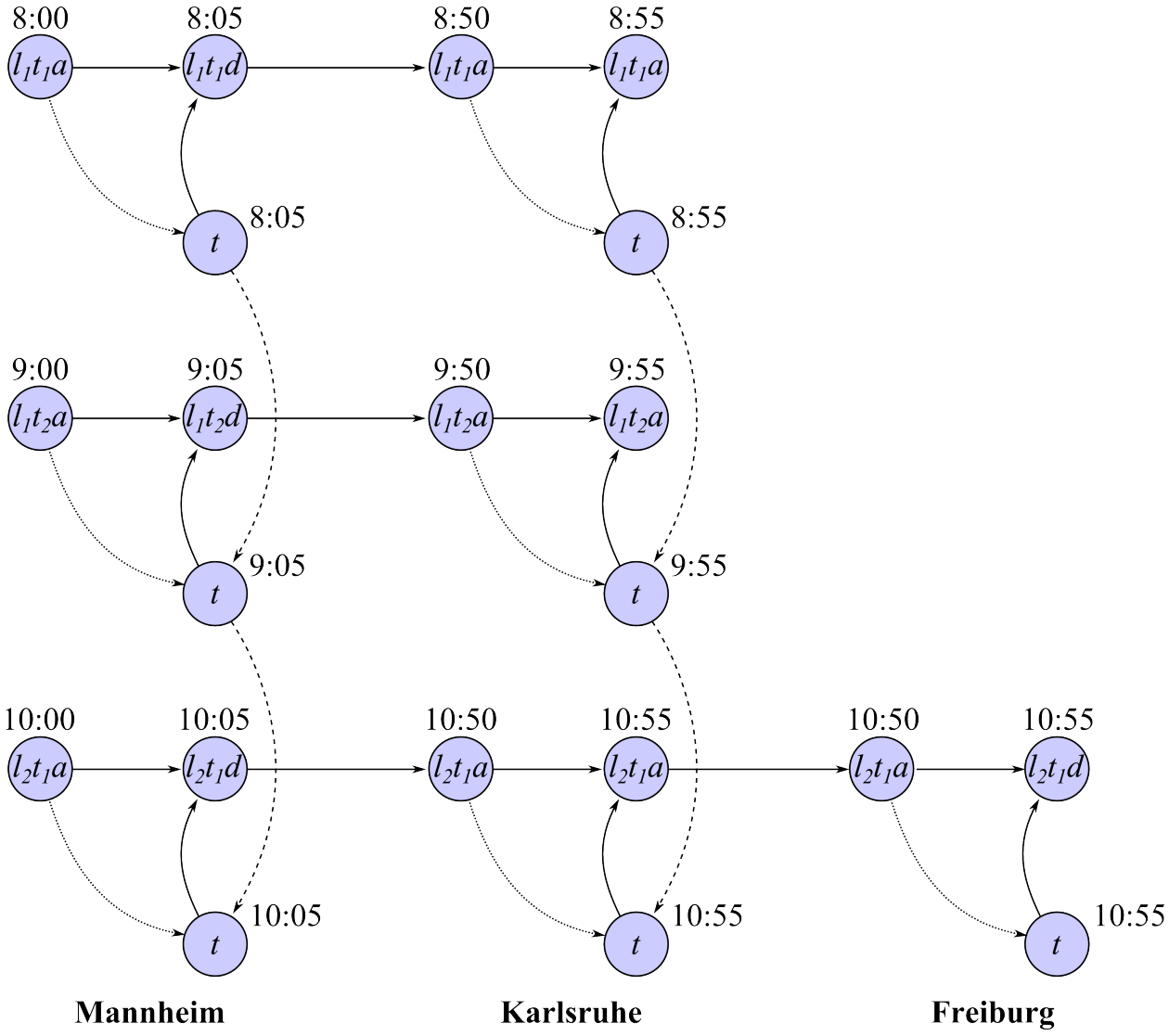

Figure 4.1: Time-expanded graph with two lines and three trips. Line l_1 has trip t_1 and t_2 and goes from Mannheim to Karlruhe. Line l_2 has trip t_1 and goes from Mannheim to Freiburg via Karlsruhe. Nodes labeled with t are transfer nodes.

of a station node have normally a fixed cost representing the minimum transfer time (transfer buffer). Arcs going from route nodes to station nodes allow to transfer to another vehicle in that station and normally have a fixed cost of zero. Fig. 4.3 illustrates the *train-route-model* model.

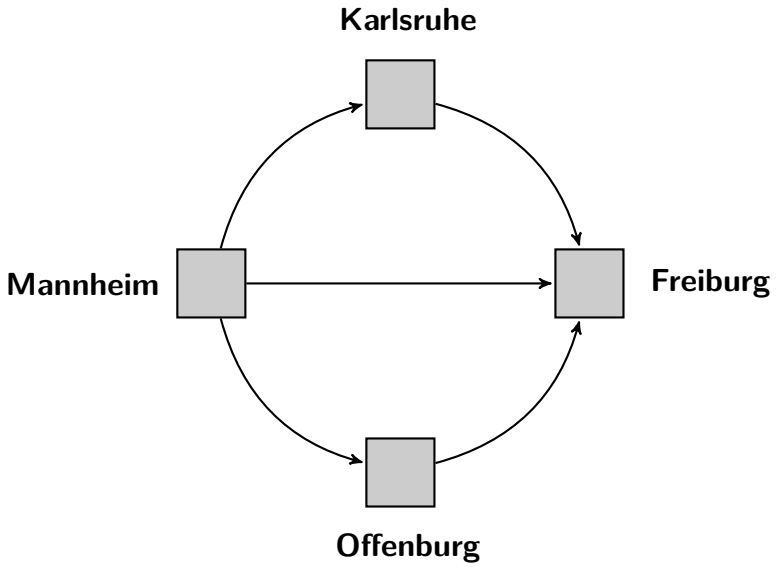

Figure 4.2: Time-dependent model with connections from Mannheim to Freiburg. These are via Karlsruhe, via Offenburg or a direct connection. Station nodes are represented as squares.

4.3 Algorithms

Both road and transit networks can be modelled as graphs. However, speed up techniques used in road networks to compute shortest paths in general do not perform well on transit networks [5]. For instance, hierarchical approaches have a positive influence in road networks when computing shortest paths. But for transit networks this is not the case as they lack of a hierarchy to exploit, especially within a city and networks with poor structure. Moreover, transit networks have to handle transfers, multiple critera and schedules, which makes the shortest path computation a more complex problem compared to road networks. In the following section we give an overview of an extension of Dijkstras algorithm to compute shortest paths with multiple criteria and a brief description of transfer patterns. More details on routing with transfer patterns are provided in Chapter 6.

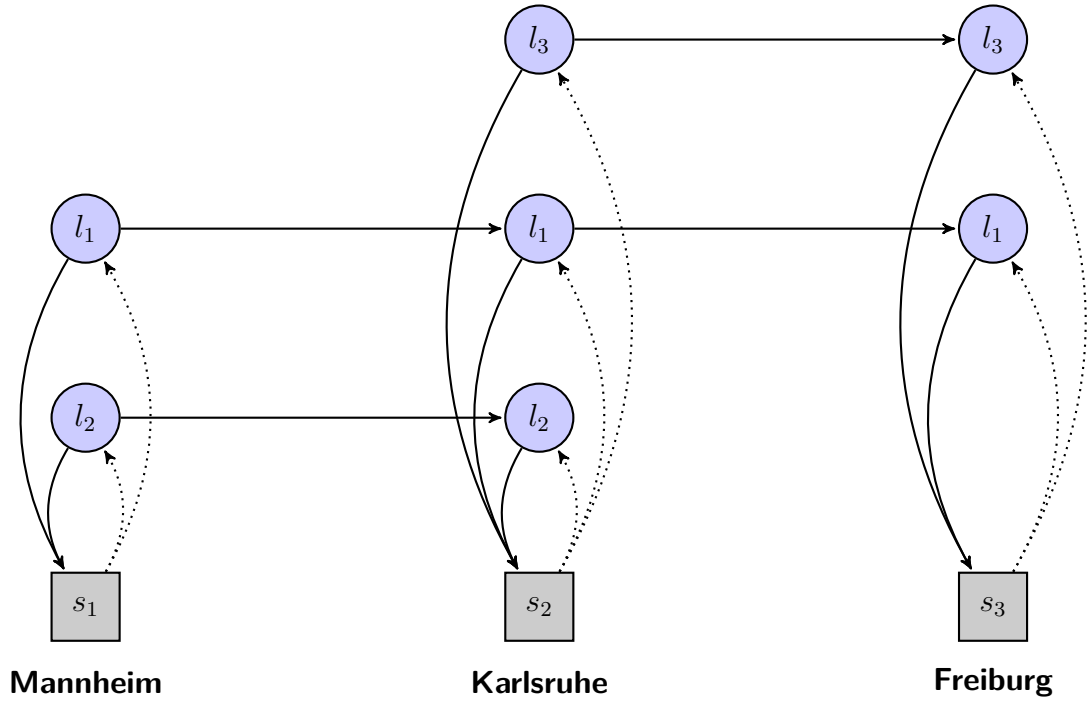

Figure 4.3: Train-route-model with three lines. Line l_1 goes from Mannheim to Freiburg via Karlsruhe. l_2 goes from Mannheim to Karlsruhe and l_3 goes from Karlsruhe to Freiburg. Line nodes are circles and station nodes are squares. Dotted arcs have cost of five minutes which is the transfer buffer. Arcs from line nodes to station nodes have cost zero.

4.3.1 Multi-label Dijkstra

Dijkstras Algorithm can be extended to minimize multiple costs and thus present to the user not just one optimal path, but a set of optimal paths. A cost is a tuple with two or more components (ie. travel time, number of transfers). Each node in the graph maintains a set of these tuples also called labels. Initially, all sets are empty except for the source node which has one label. The elements in the priority queue now are labels. As in the normal Dijkstras algorithm in each iteration an element (now a label) is taken out from the priority queue and settled. The incident arcs of the node where the label belongs are relaxed and new labels are created. If a new label is incomparable to all labels in the label set of the tail node, then the new label is inserted. Furthermore, labels no longer optimal are discarded. If one of the existing labels in the set dominates the new label, the new label is not inserted. For details on how to maintain Pareto label sets see Section 6.2

The multi-label Dijkstra can be adjusted to use it on all graph models described before. Nevertheless, the running time of this variant with multiple criteria increases by a factor of 10 compared with the normal Dijkstra with scalar costs [26].

4.3.2 Transfer Patterns

Routing with transfer patterns is a state-of-the-art algorithm for public transportation. It is based on precomputated data and allows for multi-criteria shortest paths queries within milliseconds. Routing with transfer patterns has the following basic components:

- Transfer patterns precomputation. All shortest paths between stations are computed. From these paths the set of transfer patterns is extracted. A transfer pattern denotes the sequence of stations where a transfer happens. Afterwards, they are stored in directed graphs.
- Direct-connection tables precomputation. Time tables are stored in a special data structure that allows for fast direct-connection queries at a specific time.
- Query graph construction. At query time, precomputed transfer patterns between source and target station are retrieved and a directed graph is built. A query is: from station A, how to get to station B at time t ($A@t \rightarrow B$).
- Query graph evaluation. To obtain the shortest paths at a specific time a Dijkstra on the query graph is computed. Arcs are evaluated by direct-connection queries.

Chapter 5

Combining Road and Public Transportation Networks

In the previous chapters, we gave a description of how to model road and public transportation networks as graphs, as well as algorithms to compute shortest paths on such graphs. In the following, we recall these descriptions and explain our approach to combine both networks. Finally, we describe how to join uni-modal graphs into one multi-modal graph, which will be used later on to compute transfer patterns.

5.1 Graph Model

In order to compute multi-modal routes, we build three different graphs. Each of these graphs represents a mode of transportation: Walking G_w, traveling by car G_c and using transit G_t. Once having the three graphs, they are combined into one multi-modal graph G_m.

5.1.1 Transit Graph

First, the transit graph G_t has to be modeled. Therefore, it is necessary to extract trips, stations and their corresponding schedules from a GTFS-feed. A *trip* is defined as a sequence of stations $s_0, s_1, ..., s_n, n > 0$ where a vehicle departs at a specific time from station s_0 and goes through each station consecutively until arrival in s_n. Trips that share the same sequence of stations at possibly different times and do not overtake each other belong to a *line*. Each line is stored in a table as illustrated in Table 5.1

Afterwards, the transit graph $G_t(V_t, E_t)$ is built in a time-dependent approach. This model is similar to the train-route-model but with more nodes per station. For each station i there is a station departure node $sd_i \in V_t$ and a station arrival node $sa_i \in V_t$. For each line x that goes through a station s_i there is an arrival node $l_x a_i \in V_t$ and a departure node $l_x d_i \in V_t$. Nodes in a station s_i are connected with four types of arcs: Arcs from line arrival nodes to the station arrival node $(l_x a_i, sa_i)$

Table 5.1: Times of two trips belonging to line 20. A vehicle departs at 8:00 from station s_5 and arrives at 8:05 at station s_{67}, stays there one minute and then goes through station s_{12}, finally arrives at s_{40} at 8:15.

line 20	s_5	s_{67}	s_{12}	s_{40}
trip 1	8:00	8:05 8:06	8:10 8:10	8:15
trip 2	8:40	8:45 8:46	8:50 8:50	8:55
...	...	...	...	...

(alighting a vehicle), arcs from station arrival nodes to station departure nodes (sa_i, sd_i) (staying on a station possibility to transfer), arcs from station departure nodes to line departure nodes $(sd_i, l_x d_i)$(boarding a vehicle), arcs from line arrival nodes to line departure nodes $(l_x a_i, l_x d_i)$(staying on a vehicle at a station). All of these arcs have cost zero with the exception of arcs from station departure to line departure nodes that have an additional transfer buffer. This is the time a traveler needs to change from one vehicle to another.

Additionally, there are time-dependent arcs connecting stations. These arcs go from line departure nodes on station s_i to line arrival nodes on the subsequent station s_j $(l_x d_i, l_x a_j)$ (traveling on a vehicle). These connections are also referred as *elementary connections* [20]. Fig.5.1 illustrates the graph model.

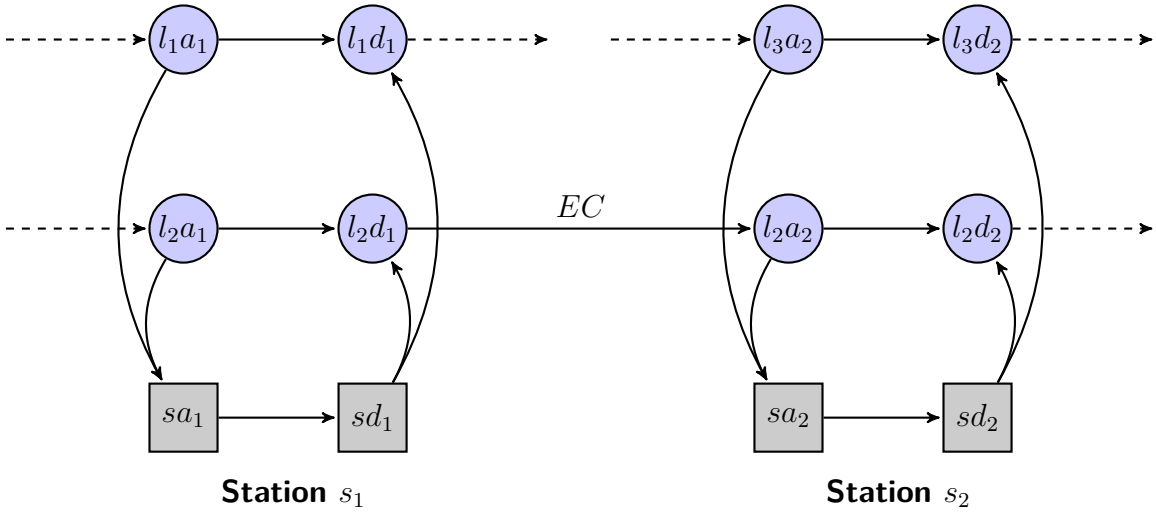

Figure 5.1: Graph model showing nodes and arcs of two stations. In this example lines l_1 and l_2 stop in station s_1. Lines l_3 and l_2 stop in station s_2. Note that line l_2 stop in both stations, the arc labeled with EC is an elementary connection between both stations.

5.1.2 Walk and Car Graphs

The walk graph G_w is a static road network, where each arc is weighted with the time (in seconds) it takes to walk its corresponding road segment with average speed of 4km/h. Remember that each intersection in the map corresponds to a node and the road segment between intersections is represented as arcs.

Like the walk graph, the car graph G_c is a road network with the same properties. The difference is on the weight of the arcs that correspond to the average travel time depending on the road type (for highways the average speed is higher as for small streets in the city). We consider speeds from 5km/h to 110km/h. For simplicity, we assume for both road networks that all roads can be traveled in both directions and no turn restrictions were considered.

5.2 Contracting Road Graphs

One way to have a more compact model is to contract the walk and car graphs with Contraction Hierarchies before joining them into G_m [21]. Because we want to use G_m to compute transfer patterns between stations, distances between stations must be preserved. However, the actual path which consists of the sequence of nodes in the road graph is not relevant, thus these nodes can be contracted. This contraction process will result in a more compact model and less visited nodes during a Dijkstra search. For a transfer pattern computation it is necessary to compute for each station one Dijkstra to everywhere, which is a very expensive computation. This contraction process reduces the search time by avoiding unnecessary propagation of results on the road graph. In chapter 6 we describe in detail these Dijkstra searches and how transfer patterns are computed.

The contraction process can be divided into two tasks. First, it is necessary to find for each station the nearest nodes in G_c and G_w. Second, all nodes in G_c and G_w are contracted except the nearest nodes which have node order infinity. To define the contraction order we use the edge difference heuristic with a fixed limit of 10. This limit helps to maintain a low number of shortcuts added. But at the same time it causes some uncontracted nodes that are not near a station. However, the number of uncontracted nodes stays low compared with the total number of nodes in the road graphs.

5.3 Joining Graphs into a Multi-Modal Graph

In the following we describe the steps to join the walk graph G_w with the transit graph G_t. The same procedure is used to join the car graph G_c. The join process can be divided into the following steps:

1. Contract the walk graph G_w with Contraction Hierarchies as explained in section 5.2. After the contraction we obtain the set of uncontracted nodes V_{un}.
2. Add all nodes in V_{un} to V_t
3. Add to E_t arcs $(u, v) \in E_w$ where $u, v \in V_{un}$. Note that these arcs could be also shortcuts added during the contraction step.
4. For each station s_i find the nearest node $nn_i \in V_{un}$ and add the following arcs:
   - Arc from station arrival to nearest node (sa_i, nn_i).
   - Arc from nearest node to station departure (nn_i, sd_i).

The combination of the walk, car and transit graph is referred as multi-modal graph G_m. Fig. 5.2 illustrates the multi-modal graph.

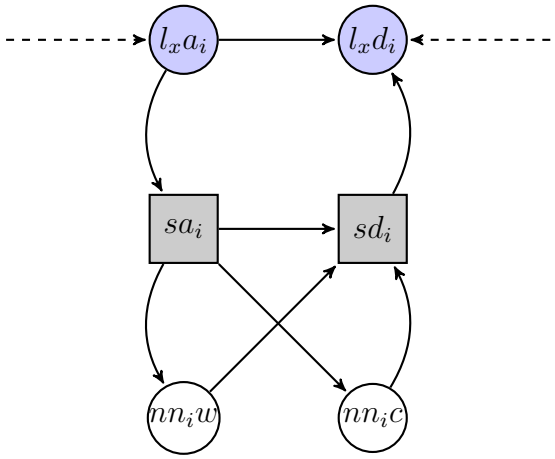

Figure 5.2: Graph model showing nodes and arcs of station s_i. In this example line l_x stops in station s_i. The nodes $nn_i w$ and $nn_i c$ are the nearest nodes in the walk and car graphs to s_i, respectively.

Chapter 6

Implementing Multi-modal Transfer Patterns

This chapter elaborates on the implementation of multi-modal transfer patterns. First, it provides a description of the selected cost model. Afterwards, it explains how to compute profile queries to obtain optimal paths between all pairs of stations. Later on, it provides an explanation on how to retrieve transfer patterns from results of profile queries and build from them a DAG. Finally, we describe a filtering technique to remove undesirable routes and provide a small and reasonable set of optimal paths for the user.

6.1 Transfer Patterns

This section is dedicated to give an overview of routing with transfer patterns. We explain the Transfer Patterns algorithm and how to include different modes of transportation. We refer on the following to the algorithms and components of the original publication on transfer patterns by Bast et al. if not stated otherwise [7].

The basic idea of the Transfer Patterns algorithm is to exploit a special property of transit networks. This is that given a source and target station there is a limited number of transfer patterns that are optimal. A transfer pattern is described as the sequence of stations on a path where a transfer happens. Having this in mind, then at query time it is only necessary to search on this patterns the best option at a specific time. Coupled with an efficient way to look up direct connections, query responses of just a few milliseconds are possible even on very large networks.

Routing with transfer patterns is based on two components. The first is the precomputation of the transfer patterns which is a time expensive process and one of the main drawbacks of the algorithm. The second component is a structure and algorithm that efficiently retrieves from time table data the next feasible direct connection between two stations at a specific time.

6.2 Cost Model

One important criterion to choose a specific route is the total travel time, but there are many other criteria that a user might want to use (price, minimum walking, minimum car usage). Especially, in public transportation the number of transfers, total travel time and monetary costs are factors considered to define the set of optimal paths. Using travel time as only criterion yields to exactly one optimal path. Namely, the path with the smallest travel time, whereas multiple criteria scenario could yield to a set of optimal paths. For instance, considering bicriteria (travel time, number of transfers) a path with 30 min travel time and no transfers (30, 0) is incomparable to a path with 20 min travel with one transfer (20, 1). Deciding which of these options is the best depends on users preferences. One user would prefer an option with no transfers and other user would prefer the fastest connection although it implies more transfers. On the other hand, there are options that are strictly less or worse than others. For example, a path with 10 minutes and no transfers (10, 0) is less than other with 30 minutes and no transfers (30, 0). The latter option should be prunned. In order to provide multiple options to the user and optimise multiple criteria we implement Pareto sets [22] which contain non-dominated costs.

In our framework a cost c is a tuple (x, y, z) where x is the total travel time relative to the start time (e.g. 30min traveled since start), also refered as *duration*. The second component y is the transfer penalty (i.e. number of transfers) and the third component z is the total travel time by car. We say cost a with components (x, y, z) is less or equal than cost b with components (x', y', z') if all components of a are less or equal than components of b:

$$(x, y, z) \leq (x', y', z') \text{ iff } (x \leq x') \wedge (y \leq y') \wedge (z \leq z')$$

Cost a is less than cost b, we say also a dominates b in the Pareto sense, if one of the following rules apply:

$$(x, y, z) < (x', y', z') \text{ iff } (x < x') \wedge (y \leq y') \wedge (z \leq z')$$
$$(x, y, z) < (x', y', z') \text{ iff } (x \leq x') \wedge (y < y') \wedge (z \leq z')$$
$$(x, y, z) < (x', y', z') \text{ iff } (x \leq x') \wedge (y \leq y') \wedge (z < z')$$

Cost a and b are incomparable if neither $(x, y, z) \leq (x', y', z')$ nor $(x', y', z') \leq (x, y, z)$.
Note that the walk duration is not a component of our costs and hence not considered as a Pareto criterion. An additional criterion implies more comparisons so that its implementation would lead to excesive precomputation times.

To obtain the optimal paths with more than one criterion, we use a multi-criteria Dijkstra[27]. This algorithm maintains a set of Pareto-optimal labels rather than a single cost value per node. Now, instead of settling a node, labels are settled. Each

label contains the tentative cost to that node and the current time. We denote the tentative cost of a label l as l_c and the current time as l_t. This time corresponds to the departure time plus the duration component of the cost. For initial labels with cost zero l_t is the same as the departure time.

6.3 Multi-label Profile Queries

We want to compute optimal transfer patterns between stations. Therefore, we need to compute for each station a multi-label Dijkstra on G_m to all other stations. In the following we call this search *profile query*. A profile query begins with initial labels at all departure events. Initial labels have cost zero on all components. The starting node can be the station arrival node or the station departure node. Starting the profile query at the station arrival node allows for initial walking/car usage, whereas starting from the station departure node walking/car usage at the beginning is not possible.

If all labels have the same departure time, it suffices to compare them in the Pareto sense. However, with different departure times, it is necessary to extend the dominance relation to preserve patterns departing at different times. A label a dominates a label b if a departs after b and a has lower or equal costs than b:

$$a(dt, c) < b(dt, c) \text{ iff } (dt_a > dt_b) \wedge (c_a \leq c_b)$$

The comparison of the costs is performed as explained in section 6.2. One example is a label that departs at 9:45 and arrives at a specific node at 10:30 with cost (45, 0, 0) dominates other label that departs at 9:30 and arrives at 10:40 with cost (70, 0, 0). Labels with the same departure time are compared in the Pareto sense.

6.4 Storing Transfer Patterns

After computing a profile query for a given station A, the settled labels at each reachable station B represent the optimal paths from station A to B. Each of these labels are backtracked by referring to the parent label to obtain the complete path. From this path we extract the sequence of stations where a transfer happens. Note that in our scenario a transfer can also be a change of transportation mode. All transfer patterns with source station A are stored in a Directed Acyclic Graph (DAG) for station A. This graph contains a *root node* representing the source station A, a *target node* for each reachable station B, and several *prefix nodes* C_i representing a station in a transfer pattern $AC_1, ..., C_nB$. Arcs in a DAG are labeled with the mode of transportation, which for our implementation are car, walk and transit. Fig.6.1 shows an example of this structure.

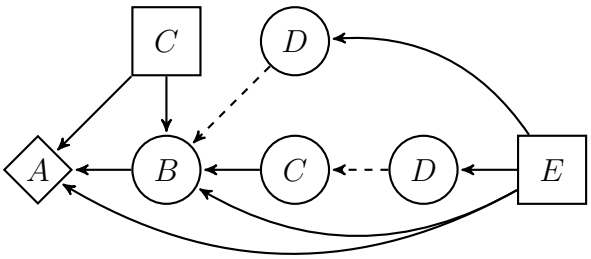

Figure 6.1: The graph represents transfer patterns AE, ABE, AC, ABC, ABDE and ABCDE. Circle nodes are prefix nodes, the diamond node is the root node and the square nodes are target nodes. Dashed lines represent walking between stations. Note that transfer patterns ABDE and ABCDE are multi-modal patterns.

6.5 Query Graph

To answer a query from station A to B at time t ($A@t \rightarrow B$), the precomputed transfer patterns between A and B must be retrieved from the DAG for station A. Based on these patterns a *query graph* is constructed, where nodes represent stations and arcs are either direct connections without any transfer in between or paths on the car/walk graph between the stations. The first step on the construction algorithm is finding the target node B in the DAG, we refer to this node as v. If v is not found, no path between A and B exists. Otherwise, it inserts a node with label B if not existing yet. The second step consists of adding to the query graph all succesor nodes of v with labels $U_1, ..., U_n$ and arcs (U_i, B). Additionally, on our implementation for each arc the mode of transportation is stored. The second step is repeated recursively for all succesors of v. Fig. 6.2 shows two query graphs built from the DAG in Fig.6.1. One for query $A@t \rightarrow C$ and the other for query $A@t \rightarrow E$.

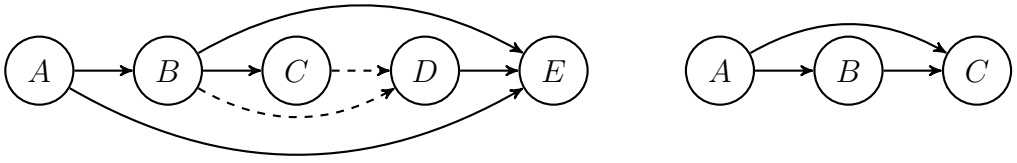

Figure 6.2: Query graphs built from DAG in Fig. 6.1. The left query graph corresponds to the query $A@t \rightarrow E$ and the right one to the query $A@t \rightarrow C$. Walking edges are dashed. Other edges are direct connections between stations.

6.6 Location-to-Location Queries

Answering location-to-location queries is a very desirable feature in realistic route planning applications. A traveler normally wants to know how to get to a specific location and not just to a station. Instead of a source and target station, a location-to-location query takes as input a source location L_s, a target location L_t and a departure time t ($L_s@t \rightarrow L_t$). The result of such a query consists of routes that do not depart before t and arrive at L_t. It includes the walking time to a set of potential source stations, which are near the source location and the walking from potential target stations, which are near the target location. We call these stations *entry stations* of the source and target, $EN(L_s)$ and $EN(L_t)$ respectively.

6.6.1 Entry Stations

To compute $EN(L_s)$ we define an heuristic based on the assumption that people tend to accept little walking at the beginning and end of a journey. Therefore, we fix an area around the source location of 400 m and declare all stations within this area as entry stations from the source via walking. Entry stations from the target are computed analogously.

To answer location-to-location queries we follow the approach introduced by Geisberger in [20], which consists on adding two extra nodes to the query graph, one for the source location L_s and one for the target location L_t. Node L_s has outgoing arcs to all nodes in $EN(L_s)$ and the node L_t has incoming arcs from each of the nodes in $EN(L_t)$. Arcs connecting the source/target to entry stations have vehicle type walk. The query graph is built in the same way as described in section 6.5 but now retrieving transfer patterns for every pair of stations in $EN(L_s)$ and $EN(L_t)$. Fig. 6.4 shows an example of a query graph considering the source and target location.

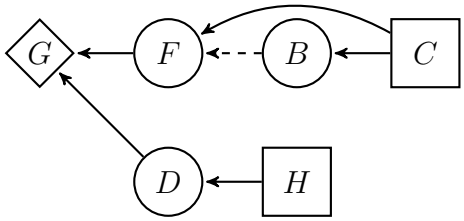

Figure 6.3: DAG for station G. Dashed lines represent walking between stations. Circle nodes are prefix nodes, the diamond node is the root node and the square nodes are target nodes.

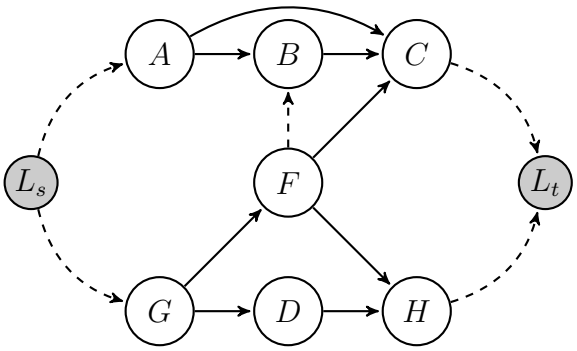

Figure 6.4: Query graph for $L_s@t \rightarrow L_t$. Entry stations to the source are A and G. Entry stations to the target are C and H. The query graph is built from DAG for station G illustrated in Fig.6.3 and from DAG for station A in Fig. 6.1. Walking arcs are dashed.

6.6.2 Evaluating the Query Graph

A query graph is evaluated by a time-dependent Dijkstra with multiple criteria [19]. During this Dijkstra search if an arc is of type walk or car, the cost is looked up in a table with precomputed durations between all pairs of stations. Walking and car durations are computed in the contracted G_w and G_c with a bidirectional Dijkstra and following a contraction order as explained in section 3.3.2. Costs of transit arcs are obtained via a direct connection query.

Direct Connection Queries Answering efficiently direct connection queries is a essential ingredient in the transfer patterns algorithm. A direct connection query answers the question *I am on station A when can I be at station B with one vehicle the earliest*. A data structure holding schedules is needed to answer this queries in an efficent way. This structure has two main components. The first one is the set of lines, where each line has a list of trips sorted by departure time. Each of these trips share the same sequence of stations and do not overtake each other. The second component is maintaining a list of incident lines for each station. Each element on the list contains a line that goes through this station and the position of the station on the sequence of stations of that particular line. For example, consider line 20 illustrated in Table 6.1 station s_7 is the second station it visits and station s_{40} is the fourth. Therefore, the incident list of station s_7 contains an element (l_{20}, 2) and the incident list of station s_{40} contains an elment (l_{20}, 4).

The algorithm to answer a direct connection query $s_i@t \rightarrow s_j$ using the described structure first intersects the incident list of s_i and s_j. The intersections results on lines that stop at s_i as well as on s_j. From these consider only the ones that stop

s_7	s_{40}
(l_{20}, 2)	(l_{13}, 2)
(l_3, 9)	(l_5, 9)
(l_6, 1)	(l_{20}, 4)
(l_6, 1)	(l_{20}, 4)
...	...

line 20	s_5	s_7	s_{12}	s_{40}
trip 1	8:00	8:05 8:06	8:10 8:10	8:15
trip 2	8:40	8:45 8:46	8:50 8:50	8:55
trip 3	9:00	9:05 9:06	9:10 9:10	9:15
trip 4	9:40	9:45 9:46	9:50 9:50	9:55
...	...	...	...	...

Table 6.1: Left table shows the incident lists of station s_7 and s_{40}. Right table shows the trip times of line 20.

first at s_i and then at station s_j. The next step is search for the first feasible trip of the line. This is the first trip that departs after t from s_i. In the example of Table 6.1 a query s_7@8:30 $\rightarrow$ s_{40} finds line 20 in both incident list of s_7 and s_{40} and the first feasible trip arrives at s_{40}@8:55.

6.7 Types and Thresholds (TNT)

The combination of the graphs G_w, G_c and G_t into G_m allows to unlimited walking and car usage. Furthermore, this combined graph with a Pareto-cost model produces a huge amount of path variations. Most of them have only a slight difference between the car usage and total travel time. One reason of the increased number of optimal paths is that each portion of transit between stations can be replaced with car usage. Traveling by car, within a city, is usually faster than using public transportation. So that replacing a portion of transit increases the car duration, but decreases the total travel time. Note that in the Pareto sense a cost dominates another cost only if it is less or equal in all components, therefore all these variations are optimal. Consider the example of using the car to the nearest bus station and then continue from there with a bus line. It is also optimal to go with the car to the second nearest station on the bus and continue from there with transit. The first option has lower car duration but more travel time, whereas the second option has greater car duration but less travel time. As mentioned in Section 6.2, the walking duration is not considered as a Pareto criterion. Therefore, a lot of Pareto-optimal paths might possess high walking portions which also have to be filtered out to get reasonable results.

In addition to the increase of optimal paths that are similar due to car usage, another problem arises. Some of the resulting paths, although Pareto-optimal, are unreasonable and thus unlikely that a user will choose them. Examples of such unreasonable paths are: "take the car 3 hours, then a train 2 minutes" or "walk one hour, take a bus 5 min, then a taxi 2 min".

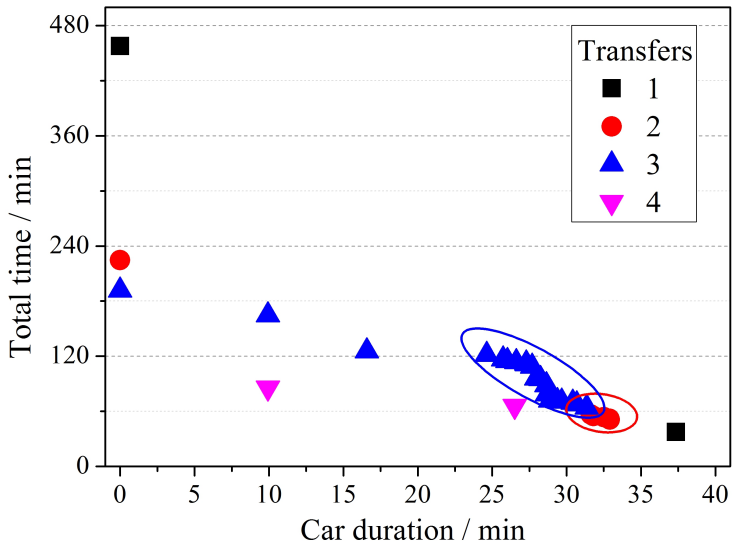

Figure 6.5: Resulting paths which differ slightly on car usage and total travel time. Two groups of very similar paths can be identified. The first composed by paths with three transfers on the range of 25 to 31 min in car usage. The second group have two transfers and are all in a range from 31 to 33 min in car usage.

Types and Thresholds(TNT) addresses the latter problems by filtering this large set of variations to a small and representative set of optimal paths. In the following section we give an overview of this concept introduced in [10].

6.7.1 Filtering by Types

A cost is classified as reasonable or unreasonable based on the *relative durations* of each mode of transportation. Relative durations of a mode m are $zero(m)$, $little(m)$ and $much(m)$. These follow the order $zero(m) < little(m) < much(m)$. The idea of including zero as relative duration is that the use of a transportation mode implies some arrangements that have to be done like calling a taxi, buying tickets, and so on. Distinguishing between *little* and *much* is important because there are some options that not many people are willing to do, for example walking for long distances. Moreover, the authors of [10] observed certain properties of each mode of transportation that could be useful to define a classification of reasonable paths. On the one hand, the car mode is expensive but fast and available everywhere. On the other hand, the walk mode is slow but cheap and it is also available everywhere. Meanwhile, the transit mode is limited to schedules and available only at stations but is a medium-fast way of traveling. After an analysis of availability, velocity and relative durations of each mode, three *types* were defined as reasonable:

1. Use the car for the whole journey. This option is normally the fastest but most expensive route.

2. Use much transit, much walking but no car. This option is normally the slowest but cheapest route.

3. Much transit, little walking and little car. This option is normally in terms of price and time in between type 1 and type 2.

For practical use relative durations have to be converted to concrete thresholds. The relative duration $zero(m)$ is zero minutes for all modes, $much(m)$ is infinity and for $little(walk)$ is 10 minutes. The $little(car)$ is computed based on the pure car duration in the following way: if the pure car duration is less than 20 minutes, $little(car)$ is zero minutes, otherwise it is the maximum between 10 minutes and 25 % of the pure car duration. This is the total travel time by car from one station to another. In our implementation we already precompute such durations for the evaluation of query graphs, so we also use them for the filtering process.

6.7.2 Discretization

One approach to alleviate the problem of having resulting paths with only a small difference on the car duration is to discretize the car duration into blocks of time. For example, a block could be ten minutes. First, the car durations component of the cost are rounded up to the next block. For instance 24 and 25 min are rounded to 30 min when taking blocks of ten minutes. After the rounding the tuples are filtered in the Pareto sense to produce a set of non-dominating costs. Afterwards, the original car durations are restored into the costs. Discretization was used in [10] to filter out costs as well. Fig. 6.5 illustrates a query with similar values for total travel time and car duration. Table 6.2 shows an extract of the same query with discretized car durations.

After a profile query, labels at the target stations are used to extract transfer patterns and build their respective DAG. After a profile query these labels are classified in types, labels which not belong to a type are removed. Remember that our cost is a triple (x, y, z) where x is travel time, y is transfer penalty and z is the car duration. We maintain an additional walk duration in the cost to use it in the filtering process. Afterwards, car durations of costs are discretized. After all the filtering process, remaining labels are used to compute transfer patterns.

Duration	Transfer penalty	Car duration	Discretized car duration
01:28:39	3	00:28:39	00:30:00
01:35:15	3	00:28:15	00:30:00
01:35:57	3	00:27:57	00:30:00
01:48:41	3	00:27:41	00:30:00
01:51:20	3	00:27:20	00:30:00
01:10:40	3	00:29:40	00:30:00
01:11:23	3	00:29:23	00:30:00
01:05:35	4	00:26:31	00:30:00
00:50:54	2	00:32:54	00:40:00
00:51:53	2	00:32:53	00:40:00
00:53:31	2	00:32:31	00:40:00

Table 6.2: Extract of tuples from a query and their discretized car durations. Shaded rows remain in the optimal path set after the discretization process.

6.7.3 Observations

Discretization eliminates very similar paths. However, there could still remain similar paths which are at the border of the time block. For example, costs with car durations 29 min and 31 min are similar. But they belong to different blocks, so both costs remain.

Filtering by types and discretization was used before only for normal queries (for a given departure time), but we use it now as a post-filtering method for profile queries. This filter process reduces the large set of optimal paths to a small and meaningful set of paths. However, because this is a post-filtering process, profile queries have to compute all possible optimal paths anyway. This profile queries take much more time than profile queries without the car mode. Nevertheless, reducing the number of labels computed by profile queries decreases the number of transfer patterns. Consequently, query graphs are smaller and hence faster query times can be expected. Results of profile query times are presented in Chapter 7.

Chapter 7

Evaluation

This chapter shows the results on real datasets with our C++ implementation. We present precomputation times, query times and number of transfer patterns generated with and without filtering. Furthermore, we evaluate their qualities in comparison to Dijkstra.

7.1 Setup

For the experiments three datasets with different size and structure were chosen. These correspond to the cities Vitoria-Gasteiz, Freiburg and Austin. The former is a small network composed mainly of a bus network with 333 stations. In the following this is abbreviated as Vitoria. The second is a more complex network with trams, buses and trains. It compromises also stations from neighbouring small cities which in total form a network of 1381 stations. The latter describes the biggest network with 2,709 stations. Table 7.1 presents properties of the aforementioned networks.

	Vitoria	**Freiburg**	**Austin**
Stations	333	1,381	2,709
Lines	40	569	228
Trips	2,733	2,328	4,852

Table 7.1: Overview of Vitoria, Freiburg and Austin networks. The number of trips corresponds to trips of one day of the week.

The walk and car graphs are constructed for all datasets extracting data from OSM files. We consider for these graphs the largest connected component. For the walk graph we choose an average speed of 4km/h, whereas for the car graph speeds range from 5km/h to 110km/h depending on the road segment type. Afterwards, these graphs are contracted as explained in Section 5.2. Transit graphs for all datasets are built from data of one day, which is extracted from a GTFS feed. Multi-modal graphs are built by combining the car, walk and transit graph according to Section 5.3. Table 7.2 shows details of these graphs.

	Vitoria	Freiburg	Austin
Transit graph			
Nodes	2,092	18,342	20,262
Arcs	3,145	31,972	32,169
Walk graph			
Nodes	10,237	110,403	357,845
Nodes (core)	372	1,121	3,780
Arcs	23,150	236,220	767,132
Car graph			
Nodes	10,237	110,403	357,845
Nodes (core)	358	1,068	3,894
Arcs	23,150	236,220	767,132
Multi-modal graph			
Nodes	2,822	20,531	27,936
Arcs	11,417	53,863	96,903

Table 7.2: Details of datasets used for the different networks. Nodes (core) are the remaining nodes after contraction.

7.2 Pre-processing Time

First, we compute profile queries only on the transit graph, then on the transit combined with the walk graph and finally on the complete multi-modal graph involving the three modes of transportation (transit, walk, car). As can be seen in Table 7.3 the profile queries which include the car mode produce a extremely high amount of labels compared with the transit only and transit and walk graph. One reason of this increment is the large amount of similar paths with small variations on the car duration. More details of this variations are presented in Section 6.7.

In the next step, we compute profile queries for several randomly selected stations and report average profile query time and the total number of labels generated in Table 7.4. For the Austin network only one sample could be calculated, whereas for Vitoria and the Freiburg network various samples are used to compute profile queries. This is due to the size and structure of Austin which creates too many labels per node and hence leads to a very high profile query time of 120 minutes for one station. Therefore, for the further evaluation only the Vitoria and Freiburg networks are taken into account.

The average profile query time of Freiburg (0.57 min) is less than the one of Vitoria (1.97 min), although the network from Freiburg is larger. One explanation can be found in the difference of lines and trips of the two cities. Vitoria has few lines (40) but many trips (300), which means that each line in general has a high

	Vitoria	Freiburg	Austin
Transit	155,175	100,517	652,282
Transit + Walk	475,721	351,907	2,013,686
Transit + Walk + Car	4,525,715	7,695,072	128,592,981

Table 7.3: Number of labels generated by random profile queries with different graphs. When adding the walk mode the number of labels approx. triples, whereas adding the car mode greatly increases the number of labels.

	Vitoria	Freiburg	Austin
Number of samples	269	87	1
Profile query time (min)	1.97	0.57	2,634.55
Number of labels	1.94M	1.44M	128.59M

Table 7.4: Average profile query times and total number of labels for the profile query. Note that calculating a profile query for Austin takes 2,634.55 minutes, so that further evaluations are done only with Vitoria and Freiburg.

frequeny. In contrast, Freiburg has 569 lines with only 2,328 trips. Note that in each profile query a lot of labels with different departure times are also preserved. Therefore, each profile query for Vitoria has to keep more labels than for Freiburg. Details of pruning rules during a profile query are found in Section 6.3.

Remember that we maintain a set of Pareto non-dominated labels per node, which in our implementation requires to compare every new label to all labels in the set. These comparisons are needed to check if the new label is inserted and remove existing labels if necessary. Therefore, the precomputation time depends mainly on the number of labels per node created and structure of the network. Fig. 7.1 shows how the profile query time grows with increasing labels and that times for Vitoria are higher than for Freiburg for the same number of labels due to the different ratios of nodes and labels of the networks.

As already shown in Table 7.3 the car mode generates a lot of variations, which are very similar or not reasonable as explained in Section 6.7. These variations are later filtered out. However, the filter is done as a post-process which means that it is nevertheless necessary to compute this huge amount of unreasonable paths. This is the main drawback of our approach. This problem could be reduced by limiting the walking and car usage or applying heuristics during the profile query to detect and prune these unreasonable paths. To apply this approach on a large network reducing the precomputation time is essential. However, for the Freiburg network the precomputation of transfer patterns for all stations takes one day, which could be considered the limit of feasibility.

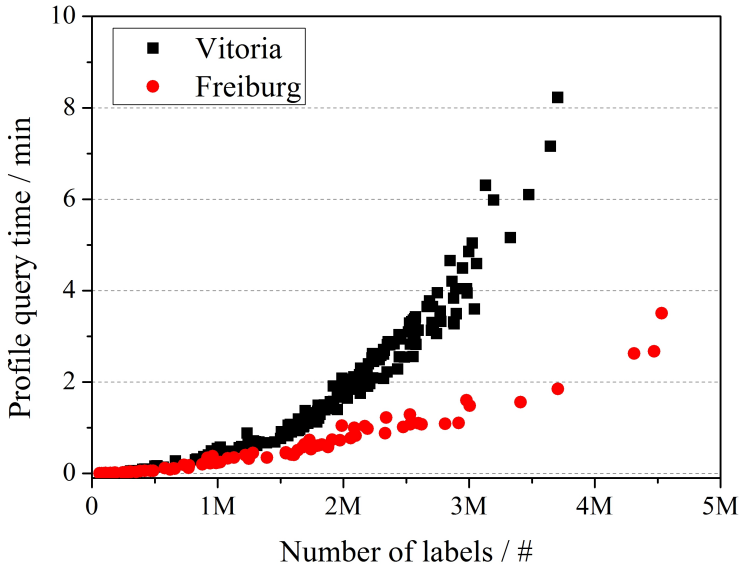

Figure 7.1: Profile query time with total number of labels. The profile query time grows with increasing labels. Furthermore, times for Vitoria are higher than for Freiburg for the same number of labels due to the different ratios of nodes and labels of the networks.

7.3 Number of Transfers Patterns

We run profile queries for randomly selected stations and register the average number of labels and transfer patterns per station pair in Table 7.5. Note that although Vitoria and Freiburg produce on average similar number of labels, the number of transfer patterns for Vitoria is significantly lower (ten times less) than the ones for Freiburg. One reason is that the Vitoria network has few lines but with high frequencies. One line with high frequencies can produce many labels which at the end compress into one transfer pattern. Freiburg has more than ten times more number of lines, so it is more likely to have different combinations, hence more transfer patterns.

Additionally, we register the number of labels and transfer patterns after the filter process. This consists of the filter by types and by discretizing of the car durations. Details of this post-process are found in Section 6.7. Results show that the filtering process reduced the number of transfer patterns by a factor of 5 for Vitoria, and a factor of 11 for Freiburg. Reducing the number of transfer patterns is especially important to obtain fast queries, as it reduces the size of the query graph. Table 7.6 shows the difference in number of nodes and arcs of query graphs built from filtered and unfiltered transfer patterns. It can be seen that a reduction of arcs of 92 % for Vitoria and 81 % for Freiburg can be obtained.

	Vitoria	Freiburg
Samples	269	87
Labels before filter	430	421.16
Labels after filter	103.93	31.92
TP before filter	14.90	144.93
TP after filter	2.86	12.82

Table 7.5: Average number of labels generated by a profile query in a station node. TP denotes the average number of transfer patterns per station pair. Values are reported before and after the filtering process.

	Vitoria	Freiburg
QG Nodes w/o filter	44.94	89.45
QG Nodes w/ filter	5.87	29.67
QG Arcs w/o filter	123.21	338.12
QG Arcs w/ filter	9.36	63.13

Table 7.6: Average number of nodes and arcs of query graphs with one source and one target station from one hundred queries of Vitoria and Freiburg.

7.4 Quality Evaluation

7.4.1 Transfer Patterns vs Dijkstra

We evaluate the quality of the results produced by our transfer patterns implementation by calculating precision and recall. We compute one thousand queries with our implementation on transfer patterns and the same queries with the normal Dijkstra on the multi-modal graph. Afterwards, precision and recall are computed with resulting paths from transfer patterns and from Dijkstra. Precision is the fraction of retrieved paths that are relevant, while recall is the fraction of relevant paths that are retrieved. For this evaluation relevant paths are the ones produced by Dijkstra. Results show that the quality of the resulting paths is preserved due to the high values for both precision and recall (see Table 7.7). Fig.7.2 and Fig.7.3 illustrate the distribution of the results.

7.4.2 Transfer Patterns with TNT vs Dijkstra

We perform a second evaluation of our transfer patterns implementation combined with Types and Thresholds and compare results with Dijkstra. As can be seen in

	Precision	Recall
Vitoria	98.3 %	96.5 %
Freiburg	99.1 %	94.0 %

Table 7.7: Average values of precision and recall on the evaluation of our implementation on Transfer Patterns against Dijkstra including the three modes walk, car and transit.

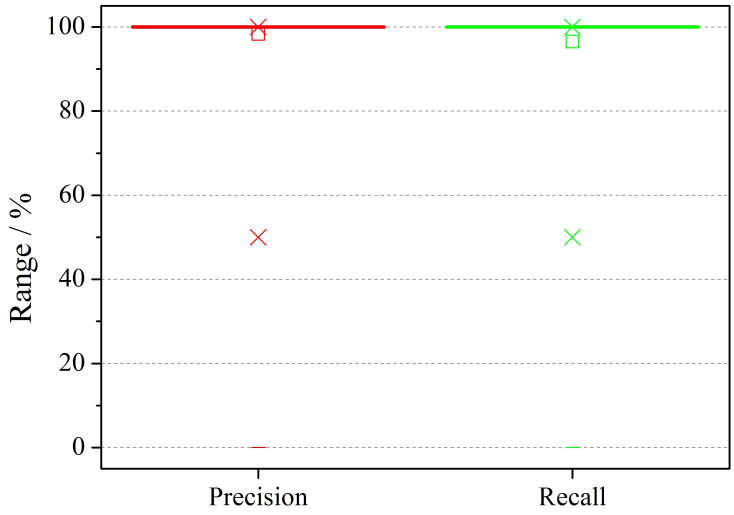

Figure 7.2: Precision and recall of unfiltered Transfer Patterns versus Dijkstra for Vitoria. Both values show almost 100 %, so that quality can be considered as preserved.

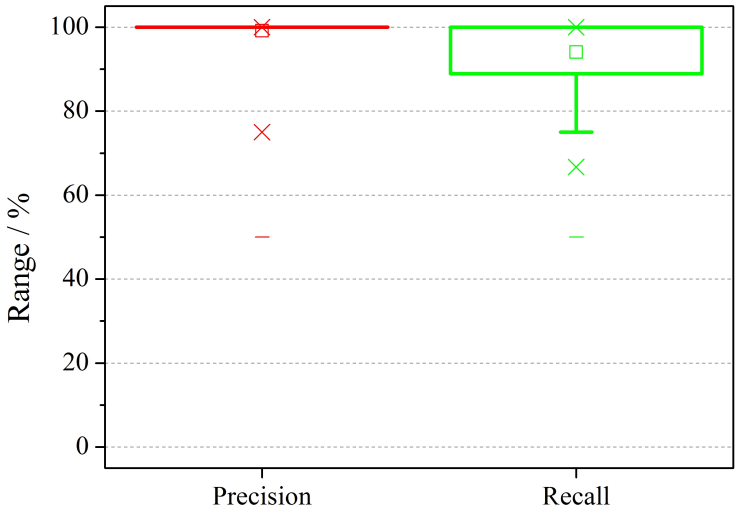

Figure 7.3: Precision and recall of unfiltered Transfer Patterns versus Dijkstra for Freiburg. Both values show almost 100 %, so that quality can be considered as preserved.

Table 7.8 precision is preserved for both datasets with values near 100 %, which is expected since the comparison between transfer patterns without filter and Dijkstra also show comparable values. However, recall values are lower for both Vitoria (34.7 %) and Freiburg (40.3 %), see Fig. 7.4 and Fig. 7.5. This decrease is due to the high number of paths classified as unreasonable and thus filtered out either by the walking or car threshold. The walking threshold is constantly set to 10 minutes, which means that higher walking durations are filtered out. As a result, in Fig. 7.6 the value in the upper left corner which represent the pure walking path is eliminated.

	Precision	**Recall**
Vitoria	93.4 %	34.7 %
Freiburg	98.9 %	40.3 %

Table 7.8: Average values of precision and recall on the evaluation of Transfer Patterns with TNT compared with Dijkstra. The evaluation includes the three modes walk, car and transit.

The car threshold depends on the pure car duration (minimum pure car duration = 20 min) and its 25% limit (see Section 6.7.1). Fig. 7.6 shows an examplary query from Freiburg. There, the pure car duration is 37 minutes which means that car usage is allowed but with a car threshold (maximum limit) of 10 minutes. However, the majority of the displayed paths with car usage are located at high car dura-

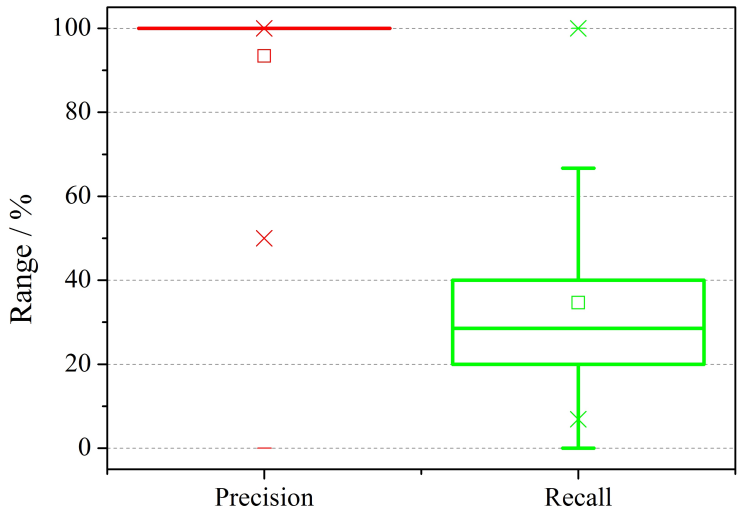

Figure 7.4: Precision and recall of Transfer Patterns with TNT versus Dijkstra for Vitoria. Precision is preserved with values near 100 %, whereas recall decreases due to the filtered paths.

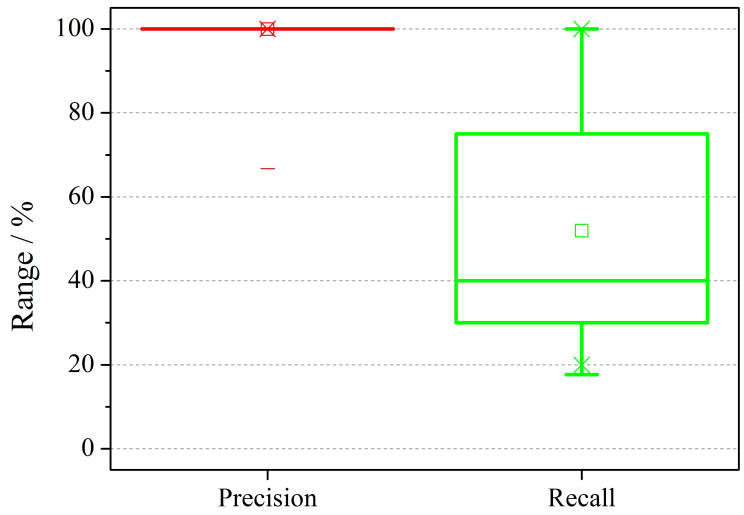

Figure 7.5: Precision and recall of Transfer Patterns with TNT versus Dijkstra for Freiburg. Precision is preserved with values near 100 %, whereas recall decreases due to the filtered paths.

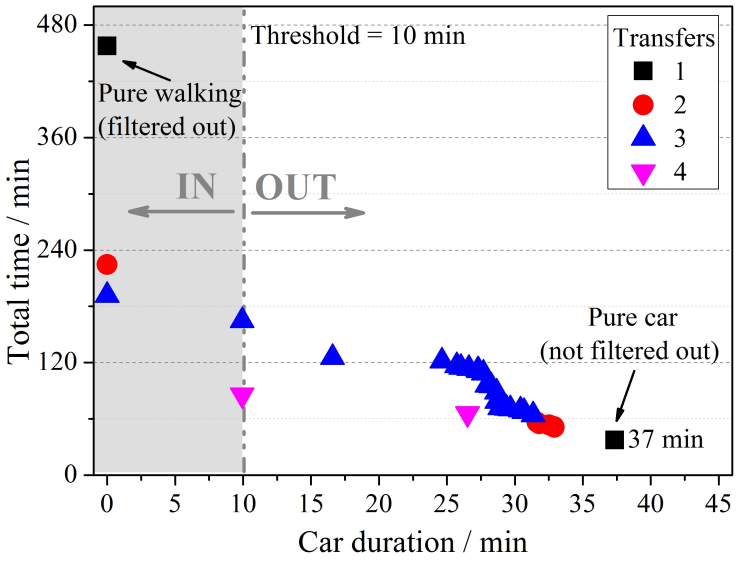

Figure 7.6: Examplary query with various different paths. The car threshold determined by the TNT filter allows only few paths with car usage. The majority of the paths are close to the pure car duration path and exceed the 10 minutes car threshold and hence are filtered out.

tions. This makes sense since the car is more flexible and faster than other means of transportation. All these options are eliminated except for the pure car duration path. This leads obviously to a low recall value. Increasing the car threshold (dotted line) to higher values less paths with car usage are filtered out and hence higher recall values are reached. Fig. 7.7 illustrates that for car thresholds which are close to the pure car duration very high recall values of up to 96.7 % for this query are achieved. There, only the aforementioned pure walking path is filtered out.

For paths with pure car duration of more than 20 minutes the recall value can be increased by changing the car threshold. However, for paths with lower pure car duration car usage is zero when applying the TNT filter. This might lead to even lower recall values. Fig. 7.8 shows both relative and cumulative frequencies of the pure car duration of one thousand queries of Freiburg. The median value is 17.6 minutes and furthermore 59.3 % of the queries possess less than 20 minutes of pure car duration. This means that for 59.3 % of the queries car usage is not allowed and hence leads to a great decrease of the recall number. This fact depends on the network size and structure. Bigger cities with larger distances exhibit a higher median of pure car duration. This can be seen in Fig. 7.9 with the Austin network, which has a median of 34.8 minutes. There, 20.3 % of the queries show lower pure car duration than the 20 minutes limit of the TNT filter, which is much less than

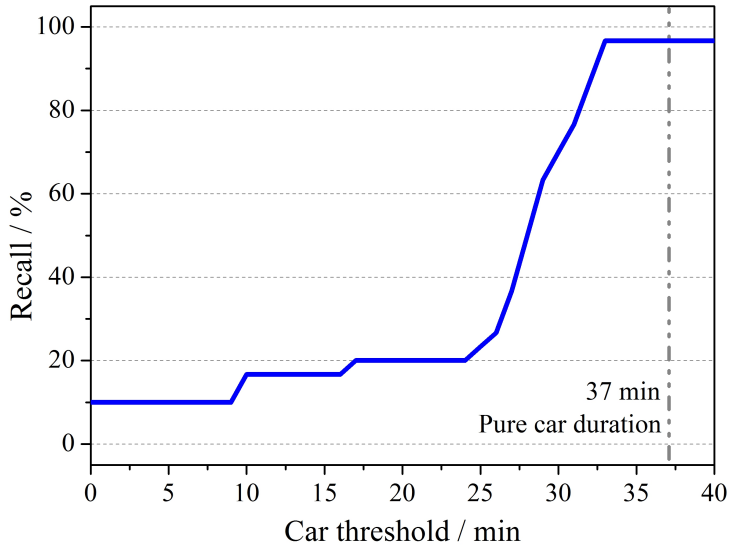

Figure 7.7: Recall values depending on the car threshold of the examplary query in Fig. 7.6. For car thresholds which are close to the pure car duration very high recall values are achieved.

for the small network of Freiburg.

The settings of the TNT filter decreases the recall values in two different ways. First, all queries with less than the 20 minutes of pure car duration are filtered out. Second, even if the pure car duration is higher and car usage is allowed a lot of path options are prohibited due to the maximum possible car duration (maximum between 25 % of pure car duration and 10 minutes). Especially for small networks these thresholds eliminate the majority of paths with car usage. This raises the question of whether these TNT settings are adequate for small cities or if they have to be adjusted to get reasonable and more paths with car usage.

7.5 Query Times

We compute location-to-location queries, choosing random locations from the walk graph and departure times in a time window of twelve hours starting at 8 am. Stations within 400 m around source and target were chosen as entry stations. If there is no station in this area, then the nearest station is taken as entry station. Table 7.9 presents average query graph size, build and evaluation time. Results show average construction and evaluation times of 10 and 14 milliseconds (ms), for Vitoria and Freiburg, respectively. Most of the total query time is consumed by the construction of the path, which includes backtracking resulting labels on the target

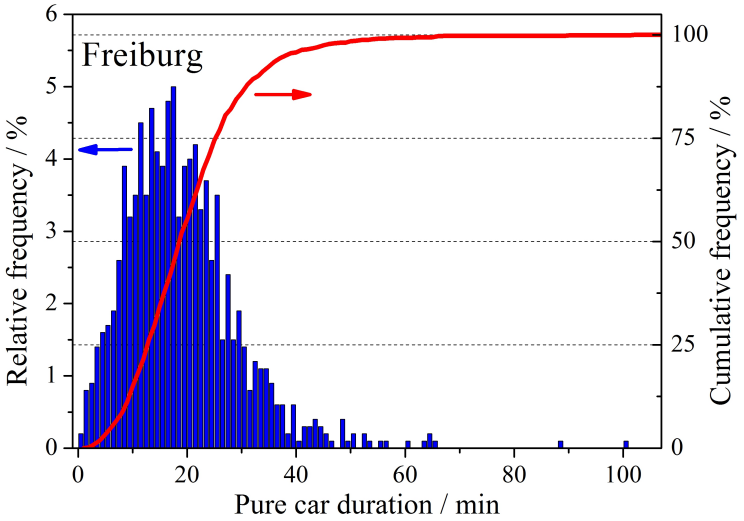

Figure 7.8: Relative and cumulative frequencies of the pure car duration of one thousand queries in Freiburg. The median value is 17.6 minutes which is lower than the 20 minutes limit of the TNT filter.

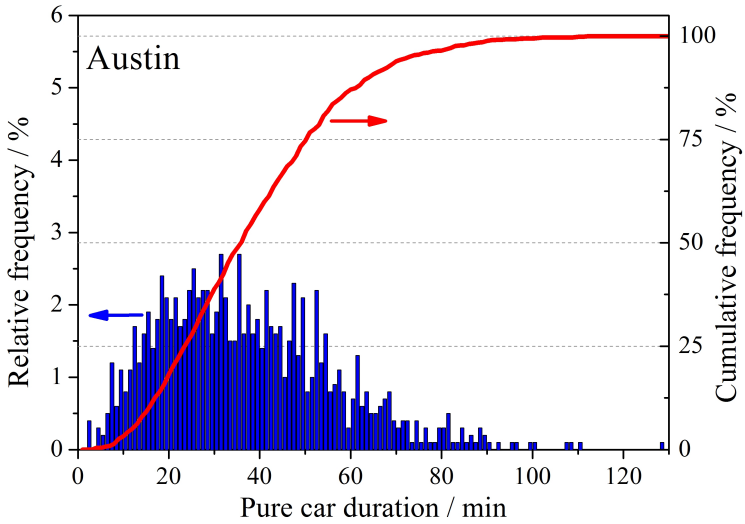

Figure 7.9: Relative and cumulative frequencies of the pure car duration of one thousand queries in Austin. The median value is 34.8 minutes which is clearly higher than the 20 minutes limit of the TNT filter.

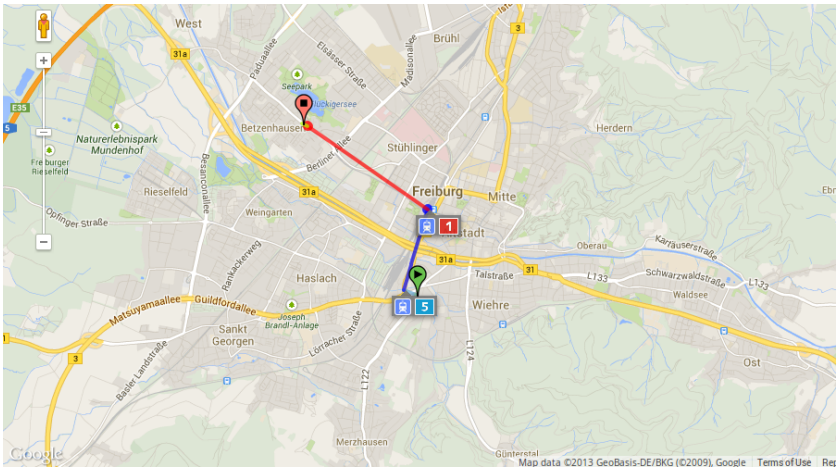

Figure 7.10: Screenshot of a route in Freiburg using transit and walking.

node and retrieving data for the client. To visualize the routes we use the java client[1] from [33]. Fig. 7.10 and Fig. 7.11 show to screenshots of routes in Freiburg, one only with transit and walking and the other including car as alternative. The rest of the build and query times are direct connection queries and look up durations. Details of construction and evaluation are provided in Section 6.5.

In previous work [6], the possibility to change the thresholds was possible and to try the query again, which we are not able to do since patterns are already precomputed with fix thresholds. However, our query times are on the order of milliseconds which implies lower computational costs for the server and less computations, because search space (query graph) is significantly lower than normal Dijkstra.

	Vitoria	**Freiburg**
Entry Stations	5.21	2.41
Nodes QG	49.03	114.46
Arcs QG	175.17	484.55
Build QG time (ms)	0.16	0.40
Evaluation QG time (ms)	2.53	4.34
Build paths time (ms)	8.17	10.09
Total query time (ms)	10.86	14.83

Table 7.9: Average query graph size, build and evaluation time. Average number of entry stations for source and target. QG denotes query graph.

[1] http://panarea.informatik.uni-freiburg.de/routeplanner

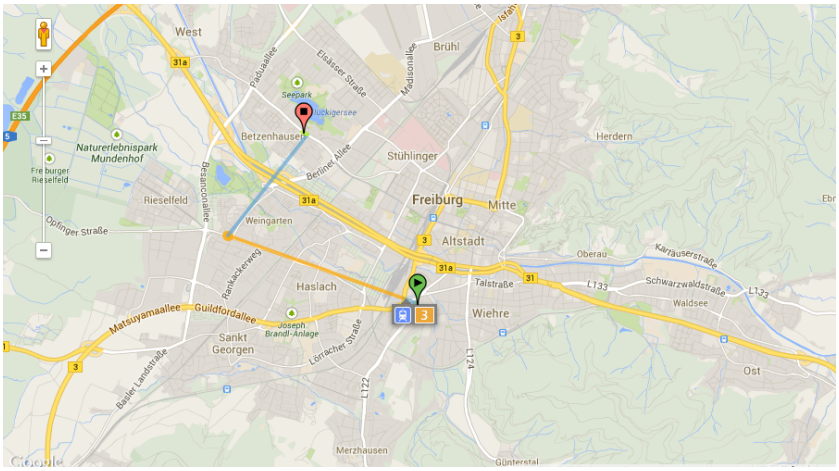

Figure 7.11: Screenshot of an alternative route in Freiburg with transit, walking and car.

Chapter 8

Conclusions and Future Work

In this work we focus on multi-modal and multi-criteria route planning with transfer patterns. A precomputation of the transfer patterns is required and at query time precomputed data is used to provide optimal routes to the user.

For the precomputation we build first a multi-modal graph comprising three graphs. Each of these graphs represents a mode of transportation. These modes are walk, car and public transportation (transit). For the latter, a time-dependent approach is followed. To have a more compact graph, road graphs (car, walk) are contracted using Contraction Hierarchies leaving uncontracted nodes that are close to a station. Uncontracted nodes are then combined with the transit graph.

The second step of the precomputation is to use the multi-modal graph to compute multi-label profile queries for each station with a Pareto-cost model. This model generates a huge amount of variations which we filtered in a post-processing step with Types and Thresholds (TNT). However, as the filter process is done after the precomputation, the computation of all these unreasonable paths have to be done anyway. For this reason, we consider our approach for large networks not feasible. This is the main drawback of our approach and one of the issues to examine in future work. The precomputation time could be reduced by limiting the walk and car usage. As we assume that a reasonable route should have a limited amount of walking/car usage, it makes sense to limit these in the graph and avoid unnecessary propagation of results on road nodes. Another way would be applying an order in the Pareto sets to reduce Pareto comparisons. Furthermore, implementing the important stations heuristic of the original publication on transfer patterns would reduce the precomputation time [7].

After the precomputation, unreasonable paths are filtered out in a post-process. This process filters paths by types and then discretizes car durations to reduce the number of paths with small differences in the car duration component. After the filter process transfer patterns are extracted and stored in directed graphs. The results indicate that using this filtering process reduces significantly the number of transfer patterns per station pair. At query time this reduction is reflected as small query graphs and thus fast query responses.

We provide an evaluation of our implementation calculating precision and recall compared to Dijkstra. This shows that quality is preserved for both precision and recall. In a second evaluation we compare Transfer Patterns with TNT filter versus Dijkstra. Here, precision is also preserved, whereas recall is significantly lower due to the selected thresholds of the filter. In future works the influence of the network size and structure on the thresholds have to be analyzed and pointed out in detail. Walk and car limits have to be adapted to each city. In our case, the used settings lead to low car usage in Freiburg.

The presented approach has fast query responses and can be used to compute location-to-location queries to serve requests from the route planner client in [33] for the Freiburg network [1].

[1] http://panarea.informatik.uni-freiburg.de/routeplanner/

Bibliography

[1] *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*. SIAM, 2005.

[2] General transit feed specification (gtfs). `https://developers.google.com/transit/gtfs/`, October 2012.

[3] Open street map (osm). `http://www.openstreetmap.org`, October 2012.

[4] Leonid Antsfeld and Toby Walsh. Finding multi-criteria optimal paths in multi-modal public transportation networks using the transit algorithm. In *Proceedings of the 19th ITS World Congress 2012*, 2012.

[5] Hannah Bast. Car or public transport - two worlds. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2009.

[6] Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result Diversity for Multi-Modal Route Planning. In Daniele Frigioni and Sebastian Stiller, editors, *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 123–136, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] 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)*, pages 290–301, 2010.

[8] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant time shortest-path queries in road networks. In *ALENEX*, 2007.

[9] Manuel Braun. Multi-modal route planning with transfer patterns. Master's thesis, University of Freiburg, December 2012.

[10] Mirko Brodesser. Multi-modal route planning. Master's thesis, University of Freiburg, April 2013.

[11] P. Fleming D. Corne, K. Deb and J. Knowles. The good of the many outhweighs the good of the one: evolutionary multiobjective optimization. In *coNNectionS 1 (1)*, pages 9–13. IEEE Neur. Net. Soc, 2003.

[12] George B. Dantzig. Linear programming and extensions. princeton university press. 1962.

[13] Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing and evaluating multimodal journeys. Technical report, Karlsruhe Institute of Technology, 2012.

[14] Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing multimodal journeys in practice. In *SEA*, pages 260–271, 2013.

[15] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multi-modal route planning by access-nodes. In *ESA, LNCS*, pages 587–598, 2009.

[16] Daniel Delling, Thomas Pajor, and Renato Fonseca F. Werneck. Round-based public transit routing. In *ALENEX*, pages 130–140, 2012.

[17] Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-constrained multimodal route plannning. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments*, 2012.

[18] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[19] Müller-Hanneman M. Schnee M. Disser, Y. Multi-criteria shortest paths in time-dependent train networks. In *WEA*, volume 5038, pages 347–361, 2008.

[20] Robert Geisberger. *Advanced Route Plannning in Transportation Networks.* PhD thesis, Karlsruhe Institute of Technologie, 2011.

[21] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[22] P. Hansen. Bricriteria path problems. In *Fandel, G., Gal, T. (eds.) Multiple Criteria Decision Making - Theory and Application*, pages 109–127, 1979.

[23] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[24] H. Ishikawa. *Global optimization using embedded graphs*. PhD thesis, New York University, 2000.

[25] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speed up dijkstra's algorithm. In *WEA*, pages 189–202, 2005.

[26] Mathias Müller-Hanneman and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *ATMOS*, pages 246–262, 2004.

[27] Mathias Müller-Hanneman and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *ATMOS*, pages 246–262, 2004.

[28] Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Timetable information: Models and algorithms. In Frank Geraets, Leo G. Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *ATMOS*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2004.

[29] Anna Sciomachen Paola Modesti. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. In *European Journal of Operational Research*, pages 495–508, 1998.

[30] 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.

[31] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *ESA*, pages 568–579, 2005.

[32] Jonas Sternisko. On compact representation and robustness of transfer patterns in public transportation routing. Master's thesis, University of Freiburg, March 2013.

[33] Niklas Meinzer Patrick Brosi Susanne Eichel, Adrian Batzill. Multimodal route planner. `http://panarea.informatik.uni-freiburg.de/routeplanner/`, 2012.

[34] Haicong Yu and Feng Lu. Advanced multi-modal routing approach for pedestrians. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 2349 –2352, april 2012.