Master Thesis - Applied Computer Science
Albert-Ludwigs-Universität Freiburg im Breisgau

Efficient Multi-Modal Route Planning

Dirk Kienle

May 7th, 2012

Albert-Ludwigs-Universität Freiburg im Breisgau
Faculty of Engineering
Supervisor Prof. Dr. Hannah Bast

Supervisor
Prof. Dr. Hannah Bast

Primary Reviewer
Prof. Dr. Hannah Bast

Secondary Reviewer
Prof. Dr. Christian Schindelhauer

Date
May 7th, 2012

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.

Freiburg, May 7th, 2012

Signature

Contents

List of Figures

List of Tables

Abstract

We present our system for computing shortest paths from A to B where we combine two types of networks: The road network and the public transportation network. The result is a multi-modal network where we are able to state a source and target address together with a date and a departure time and the system returns optimal paths that show us what roads to use and which vehicles to take.
Modeling the timetable information concerning the public transportation network, we use the time-expanded and the time-dependent approach. The main goal of this thesis is to analyze both approaches concerning the computation of the shortest paths and to evaluate the performance of both using bi-criteria optimization. Therefore, in both approaches we do not only minimize the travel time but also the number of transfers. Thus, we get several optimal solutions in order to get from A to B. For example, consider two optimal and incomparable paths where for the first we need 30 minutes and 2 transfer and for the second 45 minutes and only 1 transfer. As it depends on the customer which of the paths is preferred, we present both as optimal solutions.
Comparing the efficiency of both approaches, we also minimize in combination with the number of transfers walking between and to stations. It turns out that the time-dependent approach is about four times faster in average when we only use travel time and the number of transfers as criteria costs. But with the additional option of minimizing walking the difference becomes smaller.

Zusammenfassung

In der vorliegenden Arbeit präsentieren wir unser System zur Berechnung von kürzesten Wegen. Wir kombinieren zwei verschiedene Arten von Netzwerken: Das Straßennetzwerk sowie das Netzwerk der öffentlichen Verkehrsmittel. Das Ergebnis ist ein multimodales Netzwerk, das uns ermöglicht einen Start- und Zielpunkt sowie Datum und Zeit festzulegen und das System liefert als Ergebnis dann optimale Pfade, die uns zeigen welche Straßen und Fahrzeuge zu benutzen sind.
Wir benutzen den Time-Expanded- und den Time-Dependent-Ansatz um die Fahrplaninformationen der öffentlichen Verkehrsmittel zu modellieren. Das Ziel dieser Thesis ist es beide Ansätze bezüglich der Berechnung der kürzesten Wege zu analysieren und die Performance zu untersuchen, wobei wir nach zwei Kriterien optimieren. Daher berücksichtigen wir in beiden Ansätzen nicht nur die Reisezeit sondern auch wie oft wir umsteigen. Als Ergebnis unserer Berechnungen bekommen wir mehrere optimal Lösungen um von A nach B zu gelangen. Betrachten wir, z. B., zwei optimale und unvergleichbare Pfade, wobei wir im ersten 30 Minuten Reisezeit haben und zwei Mal umsteigen und im zweiten 45 Minuten reisen und ein Mal umsteigen. Im Allgemeinen hängt es vom Kunden ab, welchen Pfad dieser bevorzugt und daher präsentieren wir beide Pfade als optimale Lösungen.
Bezüglich des Performance-Vergleichs beider Ansätze berücksichtigen wir in Kombination mit dem Umsteigekriterium auch die Zeit, in der wir zwischen den Haltestellen laufen. Das Ergebnis unseres Vergleichs zeigt, dass der Time-Dependent-Ansatz im Durchschnitt etwa vier Mal schneller ist, vorausgesetzt wir berücksichtigen nur die Reisezeit und wie oft wir umsteigen. Minimieren wir jedoch noch zusätzlich Laufen zwischen den Haltestellen wird der Unterschied zwischen den beiden Ansätzen geringer.

Acknowledgments

I would like to thank my supervisor, Hannah Bast, who provided the topic and supported me from the start until the end whenever I needed help.

I also want to thank Mirko Brodesser and Robert Jakob for proofreading the thesis.

Dirk Kienle

1 Introduction

The problem of computing the shortest path from A to B on a road network is different than in public transportation networks. While the road network can be used at any given time we have to deal in public transportation networks with time schedules of different vehicles (bus, train, etc.). However, the basic concept is to model both networks as directed graphs. For road networks, the nodes of the graph are the junctions and the edges are the road segments. The cost of an edge is simply the time traveling across a road segment. The fastest way from A to B is then computed by applying Dijkstra's algorithm [5] to this graph.
For public transportation networks we can formulate the basic problem as follows: for a given departure station, arrival station and departure time what is the best connection to arrive as early as possible at the arrival station? There exist two approaches to model public transportation networks as directed graphs: the time-expanded and the time-dependent approach. The first constructs a directed graph where every arrival and departure event at a station is one node and the edges are either train (for a connection between two stations) or stay edges (when changing vehicles). In the time-dependent approach a node in the directed graph represents one station and an edge between two stations is a direct connection between these stations. The time we need to travel from one station to the other is computed by a cost function.
As finding optimal connections in public transportation networks relies on the time when vehicles depart from stations, it may be better to walk to the next station and take another vehicle from there. Furthermore, the start and target location of a path is usually not directly at a station. Thus, we combine both, road and public transportation network, to solve the shortest path problem. We would like to be able to state source and target address in the road network together with a date and departure time and the system should return an optimal path with respect to travel time that shows us what roads to use and which vehicles to take.
In this thesis we face the problem of computing shortest paths involving different modes of transportation which is called multi-modal route planning. The following section 1.1 gives the thesis's contributions whereas section 1.2 explains its structure.

1.1 Contributions

The thesis continues a Master Team Project where the goal was to find the shortest path concerning travel time on a combined transit (public transportation network)

and road network considering one day (Monday) of a week. Realistic features included walking between and to the next station as well as transfer buffers in order to change vehicles. Both approaches, the time-expanded as well as the time-dependent approach were implemented and evaluated concerning run time and the number of settled labels. The result stated that the time-dependent approach is approximately four times faster due to the reduced search space as those labels are ignored which do not provide a better path in terms of travel time.
The main contributions of this thesis are:

1. Realizing bi-criteria optimization for two criteria, namely travel time and the number of transfers. Optionally we minimize walking between and to stations.
2. Precomputation for the time-dependent network in order to reduce the number of labels per station and make the computation of the shortest paths much more efficient. We store for the trips of two vehicles the time when they follow the same sequence of stations up to the final station in a hash map. Then for two labels at a station where we have an entry in the hash map we compare both labels in the time-expanded sense.
3. Extending the parser to parse a schedule for a specified interval of days. For example, we are able to construct the time-expanded and time-dependent graph for the interval April 09th, 2012 to April 15th, 2012.
4. Providing analyses to compare the time-expanded and time-dependent approach concerning run time and search space and show the correspondence of both computations. In fact the sequence of settled labels in the time-dependent computation is a subsequence of those settled in the time-expanded computation. The time-dependent model is about four times faster than the time-expanded model concerning single and bi-criteria optimization (only travel time and number of transfers). However, when we minimize walking in addition the difference is smaller.

1.2 Overview

The thesis is structured as follows.

1. **Introduction** (chapter 1): The chapter provided the motivation for the thesis, gave the intuition on what tasks are to solve concerning efficient multi-modal route planning and outlined the main contributions.
2. **Related Work** (chapter 2): In this chapter we present an overview of related work. We outline what the similarities and the differences are compared to this thesis.
3. **Modeling Timetable Information** (chapter 3): This chapter provides the theory in order to understand how public transportation networks are constructed. The two approaches, the time-expanded and time-dependent approach, are explained along with their drawbacks and how to use them to model realistic networks.

4. **Implementation** (chapter 4): We explain how the time-expanded and time-dependent graph are constructed and combined with the road network. Concerning bi-criteria optimization we explain how we compute sets of optimal and incomparable paths.
5. **Evaluation** (chapter 5): In this chapter we prove that the sequence of settled labels in the time-dependent approach is a subsequence of the settled labels in the time-expanded approach. Further, we show the results of our experiments where we analyze both approaches concerning performance using single and bi-criteria optimization.
6. **Discussion** (chapter 6): We sum up our contributions and provide future work.

2 Related Work

The experimental study in [18] and [17] compares both models, the time-expanded as well as the time-dependent model, concerning several kinds of single and bi-criteria problems using datasets of Germany and France. In the time-dependent model the property of having one node per station is violated. The realistic time-dependent graph incorporating waiting time is constructed as a train route graph where for every station there is a *route-node* per train-route that passes through this station. In comparison we build the realistic time-dependent graph without the additional train route nodes, having only one node per station. We incorporate changing of vehicles into the cost functions with constant transfer time for all stations.
Concerning the time-expanded graph E. Pyrga et al [17] construct for a timetable valid for N days a fully time-expanded graph which is based on N copies of the original graph. Since the size of the fully time-expanded graph is N times the size of the original graph they simulate Dijkstra's algorithm on the original time-expanded graph. As we use plain Dijkstra we have in fact a graph N times the size of the original one.
Müller-Hannemann et al [10] use in the time-dependent model foot-edges between stations which are located close to each other. The cost of such a foot-edge is the walking time. The straight-forward modeling of foot-edges in the time-expanded case is done by time expansion. For each transfer node of a station an additional edge is maintained to the first possible transfer node at a neighbor station. In comparison to that we combine road network and transit network by adding edges between the geographically nearest nodes of both networks. Thus, we are able to model walking between and to stations.
E. Pyrga et al [17] consider bi-criteria optimization problems with the earliest arrival (EA) and the minimum number of transfers (MNT) as two criteria. They investigate three problem variants: (1) finding all Pareto-optimal solutions which is the problem considered in this thesis (optionally we also minimize walking between and to stations); (2) finding the solution that minimizes one criterion while retaining the second below a given threshold; (3) finding the lexicographically first Pareto-optimal solution (e.g., find among all connections that minimize EA the one with minimum number of transfers).
Computing all Pareto-optimal solutions leads to solutions which are not attractive to customers in real world scenarios. In [9] efficient algorithms for timetable information in public transportation systems under multiple objectives like travel time, fare and the number of transfers are considered. The challenge is to find all connections which are potentially attractive for customers and leaving out those which

are practically useless. Using bi-criteria optimization we present all Pareto-optimal solutions.
The experiments in [18], [17] and [10] show that the time-dependent approach is clearly superior with respect to performance when the original version (transfer time between vehicles at a station is negligible) of the models is considered. When considering extensions of the models (transfer buffer at stations is greater than zero and bi-criteria optimization concerning travel time and number of transfers is used), the time-dependent approach still performs better, but with a much smaller difference. Comparing both approaches in this thesis we observe that using single and bi-criteria (travel time and number of transfers) optimization the time-dependent approach is about four times faster concerning run time. But when we minimize walking between and to stations the difference is smaller; the time-dependent approach is then only about 1.5 times faster in average.

3 Modeling Timetable Information

This chapter gives the foundations of the thesis. It explains the models we use to model timetables as directed graphs. We introduce the basic models first. Then we extend them incorporating transfer buffers and traffic days and explain why including realistic features into the time-dependent model leads to problems when computing the shortest paths. As we are not only interested in finding the fastest connection but also minimize the number of transfers, for example, we explain how to compute optimal solutions under several criteria costs.

3.1 Data

A *timetable* [10] has information about the stations, stops, etc. where trains, buses or other vehicles arrive and depart from. Formally the timetable consists of a set of trains $\mathcal{Z}$, a set of stations $\mathcal{B}$ and a set of elementary connections $\mathcal{C}$ whose elements c are 5-tuples of the form $c = (Z, S_1, S_2, t_d, t_a)$. Such a 5-tuple represents an elementary connection meaning a train Z departs from S_1 at time t_d and arrives at station S_2 at time t_a without a stop in-between. If x is a field in the tuple, then the notation $x(c)$ specifies the value of x in the elementary connection c. The length of an elementary connection c is simply $t_a - t_d$. It is denoted by $length(c)$.
A timetable specifies on which days a vehicle operates. Therefore the vehicle is assigned a bit-field of N bits, assuming the timetable is valid for N days.
Transferring from one vehicle to another is only possible at a station $S \in \mathcal{B}$ if the time between arriving at S and departing from S is larger or equal to a given transfer time $transfer(S)$.

Consistent Connections For a sequence of elementary connections $P = (c_1, \ldots, c_k)$ with departure times $dep_i(P)$ and arrival times $arr_i(P)$ for each elementary connection c_i, where $1 \leq i \leq k$, P is called consistent from station $A = S_1(c_1)$ to station $B = S_2(c_k)$ if the time values $dep_i(P)$ and $arr_i(P)$ correspond to t_d and t_a and the two consistency conditions are fulfilled:

1. The departure station of c_{i+1} is the arrival station of c_i.
2. The minimum transfer durations are respected; either $Z(c_{i+1}) = Z(c_i)$ or $dep_{i+1}(P) - arr_i(P) \geq transfer(S_2(c_i))$.

3.2 The Time-Expanded Model

The time-expanded model [10] is a directed graph $G = (V, E)$ (Figure 3.1) where every time event (arrival or departure) at a station is represented as one node. The edge between the departure node of a station A at time t and the arrival node of another station B at time t' is then an elementary connection between these two stations. The cost of this so called *train-edge* is simply the travel time $t' - t$ needed going from A to B. The nodes belonging to a station A are ordered according to their time values. Then there are so called *stay-edges* (v_i, v_{i+1}) for all nodes $v_1, \ldots, v_k$ of A, where $1 \leq i \leq k - 1$ holds. These *stay-edges* represent waiting in a station and the cost for such an edge (v, w) is simply $t_w - t_v$. Thus, taking a train [12] to go from one station to another is a path made of elementary connections using the *train-edges* whereas changing the train at a station the *stay-edges* are used.

3.3 The Time-Dependent Model

The time-dependent model [10] is a directed graph $G = (V, E)$ (Figure 3.1) where every station is represented as one node. If there is an elementary connection from station A to station B then there is an edge from the corresponding nodes A to B in the graph. However, edges can only be used at certain times and the cost of an edge depends on the vehicle departing at a certain time. In order to model this fact, every edge has a cost function that takes a time and returns the earliest possible arrival time at the neighbor station. For the cost function $f_{(v,w)}(t) = t'$, where t is the departure time at v and t' with $t' \geq t$ is the earliest possible arrival time at w, the cost of the edge (v, w) is simply $f_{(v,w)}(t) - t$.
The time dependent-model is based on the assumption that for two stations A and B there are no two vehicles leaving A and arriving at B such that the vehicle that leaves A second arrives first at B.

3.4 The Earliest Arrival Problem (EA)

The earliest arrival problem [10] is defined as follows: for a given query (A, B, t_0) where A is a departure station, B an arrival station and t_0 the departure time at A, find a path that minimizes the difference between the arrival time at B and the given departure time t_0. Note that a transfer from one vehicle to another can be performed in zero time.
In order to solve the earliest arrival problem in the time-dependent model Dijkstra's algorithm has to be modified. Dijkstra's algorithm [5] computes for a given source and target node the shortest path between these nodes on a given graph. The algorithm on page 16 illustrates Dijkstra's algorithm in pseudo code. Each node $v \in V$ is assigned with a tentative cost $dist_s(v)$ stating the distance from the source

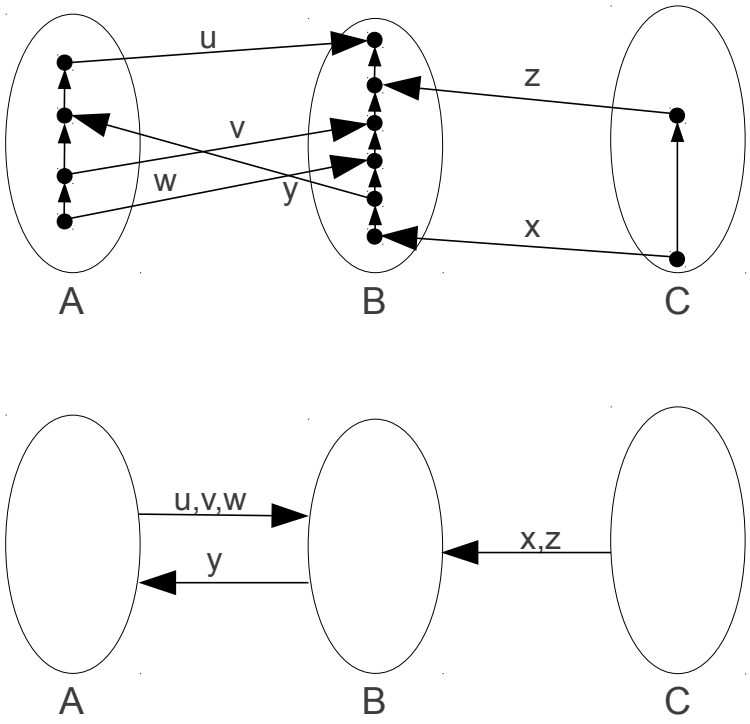

Figure 3.1: The time-expanded and the time-dependent digraph. The time-expanded (top) and the time-dependent digraph (bottom) [17] (modified) of a timetable with three stations A, B, C. There are three vehicles that connect A with B (elementary connections u, v, w), one vehicle from C via B to A (x, y), and one vehicle from C to B (z).

node s whereas $pre(v)$ sets the preceding node of v. At the beginning, for all nodes $v \in V$ we have the tentative cost $dist_s(v) = \infty$ and $pre(v) = null$. When a node has been extracted from the priority queue then it is settled. Note that a settled node never gets reinserted into the queue, i.e. each node is settled at most once. Further, an edge which is touched by the algorithm is called a relaxed edge. After the algorithm terminates, $dist_s(t)$ contains the length of the shortest path from s to t and the path can be constructed by walking along the pre-labels beginning at t until s is reached.
The algorithm is now modified such that the tentative costs are times of day. For the source station s we have at the beginning the start time $t[s]$ at that station. The tentative cost of all other stations u are set to $t[u] = \infty$. The notation $t[u]$ denotes the earliest arrival time at the station u. When relaxing an edge (u, v), $t'[v]$ of v is computed by finding the first train with time t_d that departs later than or equal to $t[u]$ at the station u such that $t'[v] = f_{(u,v)}(t_d)$. If $t'[v]$ improves the previous $t[v]$ of v then we store $t'[v]$. The length of the edge $e = (u, v)$, denoted as $\ell_e(t)$, is

then the time to wait for the departure at u plus the travel time to get from u to v. Consequently, $f_e(t) = t + \ell_e(t)$. The modified Dijkstra algorithm then proceeds with the station that has the smallest arrival time and stops when the target station is reached. The algorithm is correct as the cost function f is non-decreasing such that $t \leq t' \Longrightarrow f(t) \leq f(t')$ holds and has non-negative delay such that $\forall t, f(t) \geq t$. Concerning the time-expanded model, the earliest arrival problem can be solved by using ordinary Dijkstra as it is an ordinary graph that has edges with non-negative costs.

Algorithm 3.1 Dijkstra's algorithm

```
1: Data: A weighted graph G = (V, E), s ∈ V and t ∈ V
2: Result: Length of the shortest path from s to t
3:
4: Q ← a priority queue of nodes
5: dist_s(s) ← 0;
6: Q.insert(s, 0);
7:
8: forall v ∈ V \ {s} do
9:     dist_s(v) ← ∞;
10:    pre(v) ← null;
11:    Q.insert(v,∞);
12: endfor;
13:
14: while not Q.isEmpty() do
15:    v ← Q.dequeue();
16:    if v = t then
17:        stop ;                          // shortest path found
18:    endif;
19:    forall outgoing edges e = (v, w) do
20:        if dist_s(v) + c(v, w) < dist_s(w) then
21:            Q.insert(w, dist_s(v) + c(v, w));
22:            pre(w) ← v ;
23:        endif;
24:    endfor;
25: endwhile;
26: stop;                                  // no shortest path found
```

3.5 Comparing both Models

Consider an edge e [10], where we assume the scenario of the earliest arrival problem (see section 3.4), from a station A to another station B in the time-expanded graph. If this edge e has already been processed, all other edges e' that lead from

A to B cannot improve the path through e. Therefore, these edges do not have to be considered anymore. This fact is somehow implemented in the time-dependent model. Hence, considering a query (A, B, t_0), where A is the start station, B the target station and t_0 the departure time at A, a Dijkstra computation on the time-expanded graph relaxes the same edges in the same order as the computation on the time-dependent graph. In addition the *stay-edges* to some nodes and the edges leaving these nodes have to be considered in the time-expanded model. However in the time-dependent model the length of an edge must be computed first, which consumes run time as well.

3.6 Transfer Buffers

So far, both models do not include waiting time. But one needs a minimal amount of time to transfer between two vehicles. Modeling waiting time forces to distinguish two facts: (1) staying on a vehicle at a station involves no waiting time and (2) changing the vehicle requires a minimal amount of time.

3.6.1 Time-Expanded-Model

Based on the time-expanded graph [10], the realistic time-expanded graph, incorporating waiting time, is constructed as follows. For each arrival node at a station we introduce a new node and call it *transfer node*. The *stay-edges* are now between the transfer nodes. For each arrival node with time t we have an edge to the departure node holding the departure time of the same vehicle and a second edge to the transfer node with time $t' = t + TB$ where TB is the transfer buffer needed to change vehicles. Further, for each transfer node we have an edge to the next departure node in time. Figure 3.2 illustrates the realistic time-expanded graph.

3.6.2 Time-Dependent Model

The time-dependent model [19] can be extended using information on the routes vehicles follow. In the following the construction of a *train-route graph* is described. A sequence of stations forms a *train-route* if a vehicle starts its journey from the first node of the sequence visiting all stations in the sequence consecutively. Vehicles that visit the same sequence of stations in the same order belong to the same *train-route*. As before every node represents one station in the graph, but in addition for every station there is a *route-node* per route that passes through this station. There are edges from each station node to the route-nodes of that station modeling changing a vehicle and having the transfer time as cost. Then there are edges from each route-node to the station node modeling getting off the vehicle and having zero cost. Modeling the trips of the vehicles, there are the corresponding edges for each

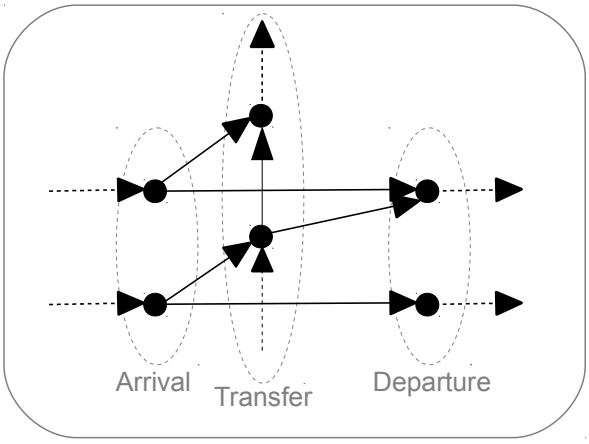

Figure 3.2: Realistic time-expanded graph incorporating waiting time. The figure shows a station with two arrival nodes connected to the corresponding departure (for staying in the vehicle) and transfer (for transferring into another vehicle) nodes.

train-route connecting the route-nodes and having the cost assigned by the cost function as described in the basic modeling.
In this thesis we implement the time-dependent model (see subsection 4.2.2) to handle transfer buffers in the cost functions. Thus, we take the time and the trip id of the arriving vehicle as input such that we have cost functions of the form $f_{(u,v)}(t, \ell)$, where t is the time we arrive at u and ℓ is the id of the trip over which we arrive at u. The cost functions then return the earliest arrival time at v respecting the transfer buffer when changing a vehicle.

3.6.3 Time-Dependent Problem

Building the transfer buffers into the cost functions to solve the earliest arrival problem with waiting time greater than zero leads to the following problem. Dijkstra's algorithm will not always compute the correct shortest path. In the time-dependent model it does not always hold that a prefix of a shortest path is also a shortest path. That means it is sometimes reasonable to arrive at a station later in time to get a faster connection to reach the target earlier in time or in some cases to reach the target at all. This problem can be solved using a multi-label Dijkstra (see subsection 3.9.2) computation on the graph. That means to encounter the problem of loosing faster connections more than one path is pursued. Therefore, for two different arrival times t and t' at a station with different vehicles, where $\mid t - t' \mid$ is smaller than the transfer buffer, both times are kept. Consider Figure 3.3 where we arrive at station C with vehicle 1 at 10:22 and vehicle 2 at 10:25. We set the transfer buffer TB to 5 minutes such that $10:25 - 10:22 < TB$ holds. Suppose we have station D as the target station. Then if we only keep the earliest arrival time

for a station in the Dijkstra computation, which is 10:22 for station C, we will never reach the target station D. In case the arrival time for vehicle 2 at C was 10:27, we would have enough time to transfer into vehicle 2 when arriving at C with vehicle 1 and would reach D.

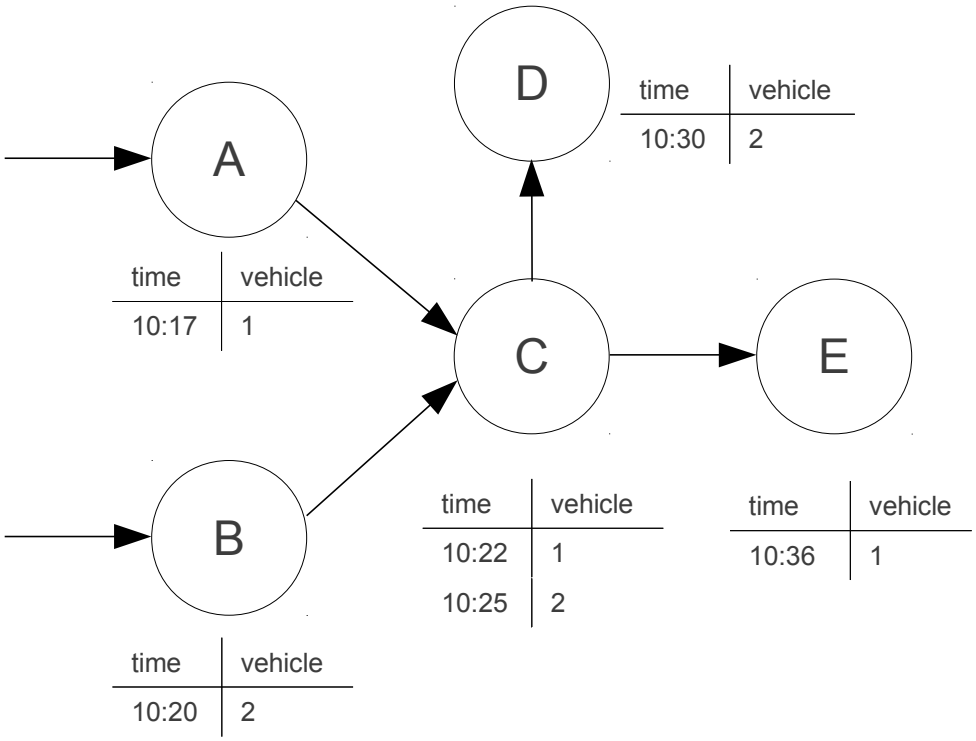

Figure 3.3: Time-dependent problem when having transfer buffer greater than zero

3.7 Traffic Days

For both models, described so far, we assumed that the timetable is identical every day. As in practice this is not the case, different traffic days must be integrated into both models. We construct a time-expanded graph where for each day there are the corresponding arrival, transfer and departure nodes together with the corresponding edges at stations A and B if and only if there is a valid elementary connection from A to B. The times of these nodes are relative to the begin of the interval the graph is constructed for. The advantage of this approach is that we can use ordinary Dijkstra to compute the shortest path but the graph grows by a factor of the number of days used.
The time-dependent model handles traffic days in its cost functions. Similar to the

time-expanded model there is only a connection between two stations for a day if and only if there is a valid elementary connection between these station. For a valid elementary connection between two stations A and B the departure time at A and the arrival time at B are relative to the begin of the interval.

3.8 The Minimum Number of Transfers Problem (MNT)

For a given query [10] consisting only of the departure station A and the arrival station B the goal is to minimize the number of vehicle transfers. The realistic time-expanded graph can be used to solve this problem by assigning all incoming edges of transfer nodes the length one and all other edges length zero. In the time-dependent case we use the cost functions (as we know the trips of vehicles over which we arrive at stations) to determine whether we change a vehicle and return one for a transfer and zero otherwise.
A shortest path in the time-expanded or the time-dependent graph from a departure node, respectively departure station, to a arrival node, respectively arrival station, is then a solution to the minimum number of transfers problem.

3.9 Multi-criteria Optimization

In the previous sections the focus was on finding the fastest connections. The costs were just scalar numbers, namely the travel time. But a customer may be interested in a connection where the number of transfers are as small as possible, walking between stations is minimized or where the price is of interest, for example. Thus, we want to find the optimal connections under several criteria. Computing optimal connections under multiple criteria [10] reduces to the multi-criteria shortest path problem where the aim is to find all Pareto optima. Pareto optimality [11] is defined as follows. Consider two d-dimensional real-value vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. If $x \neq y$ and $x_i \leq y_i$ for all $1 \leq i \leq d$, respectively if $x \leq y$ and $x_i < y_i$ for at least one i, then x dominates y and we write $x < y$. For a finite set X of d-dimensional vectors, $x \in X$ is Pareto-optimal in X if there is no $y \in X$ that dominates x.
Considering an example let us take two criteria costs: travel time and the number of transfers. With two criteria, we can get more than one optimal solution. These solutions are incomparable, which means that, for example, three hours of travel time and zero transfers is incomparable to two hours and one transfer. However there exists solutions that are strictly better: two hours of travel time with one transfer is strictly better than three hours and two transfers.

3.9.1 Size of the Pareto Set

Multi-criteria optimization [10] problems are usually NP-hard because the Pareto set is typically exponential in size. However, finding all Pareto-optimal solutions, respectively connections, may be efficient under certain circumstances. That means if the range of valid values of some criterion has an upper bound, then adding this criterion cannot lead to an exponential blow-up. Consider the number of transfers as such a criterion. In practice, there are no more than k transfers, where k is a small constant. Hence, adding the number of transfers as an additional criterion to one or more other criteria, the size of the Pareto set can only increase by a factor of k.

3.9.2 Multi Label Dijkstra

In order to compute [10][7] all Pareto-optimal solutions, we can use a variant of Dijkstra's algorithm. The algorithm on page 22 illustrates the Multi Label Dijkstra in pseudo code. In comparison to the ordinary Dijkstra algorithm, for each node in the graph we have a set of d-dimensional labels. Note that for $d = 1$, for each node we have a label that contains a single scalar value which is the tentative cost of this node. Initially there is one label at the source node. The priority queue now maintains d-dimensional labels according to some order on the set of possible labels. In each iteration the lexicographically smallest label is popped from the priority queue and settled. The outgoing edges of the node which the label belongs to are relaxed and new labels are created at the adjacent nodes. Thus, at an adjacent node w a new d-dimensional label is created and compared to all the labels in the label set of w. In case the new label dominates existing labels or is incomparable to all labels in the label set of w, it is inserted and the existing dominated labels are deleted. Otherwise, when the new label is dominated by some other label it is not inserted.

3.10 MNT combined with EA

The Multi Label Dijkstra, presented in subsection 3.9.2, can be used to solve the earliest arrival problem in combination with the minimum number of transfers problem. This is a bi-criteria problem [17] where the goal is to find all Pareto optimal solutions. The priority queue now maintains 2-dimensional labels of the form (x, y) where x is the travel time and y is the number of transfers. The lexicographic comparison defined on these labels is a refinement of the partial order which means that if $(x, y) < (x', y')$ then (x, y) must be processed first. Concerning the time-expanded model the cost of an edge is then not only the travel time but the cost pair $(x, y) = (EA(e), MNT(e))$, where $EA(e)$ is the cost of e when solving the realistic version of the EA problem and $MNT(e)$ is the cost of e when solving the MNT

Algorithm 3.2 Multi Label Dijkstra

```
1: Data: A weighted graph G = (V, E), s ∈ V and t ∈ V
2: Result: The set of all Pareto-optimal solutions at t
3:
4: Q ← a priority queue of labels of the form x = (x_1, ..., x_d)
5: Q.insert(s,x_0);                                   // insert the zero label
6:
7: while not Q.isEmpty() do
8:     (v,x_v) ← Q.dequeue();
9:
10:    forall outgoing edges e = (v, w) do
11:        y_w ←x_v +x_e;                             // create new temporary label
12:        insert ←true;
13:
14:        forall labels x_w at w do
15:            if x_w ≤ y_w then                      // x_w dominates y_w
16:                insert←false;
17:            endif;
18:            else if x_w ≥ y_w then                 // y_w dominates x_w
19:                removeLabel(w,x_w);
20:            endif;
21:        endfor;
22:
23:        if insert is true then
24:            insertLabel(w,y_w);                    // insert y_w into the label set of w
25:            Q.insert(w,y_w);
26:        endif;
27:    endfor;
28: endwhile;
```

problem. With the canonical addition, i.e. $(x, y) + (x', y') = (x + x', y + y')$, and the lexicographic comparison, i.e. $(x, y) < (x', y') \iff (x < x') \vee (x = x' \wedge y < y')$ defined on these cost pairs and the start-node initialized to *(0,0)*, for each node of the destination station we can compute a shortest path. That gives the set of all Pareto-optimal solutions at the destination station.
In the time-dependent model the cost of an edge determined by the cost function $f : (\mathrm{T}, \mathbb{N}) \to (\mathrm{T}, \mathbb{N})$, where T is a set representing time and $\mathbb{N}$ are nonnegative integers, is a pair (x, y) with x being travel time and y being the number of transfers. The lexicographic comparison and the canonical addition to these pairs is defined exactly as for the time-expanded model. The problem that arises here, similar to the problem we had with transfer buffers, is that labels computed along prefixes of shortest paths do not necessarily belong to shortest paths any more. Hence, it does not suffice to keep all labels whose time difference is smaller or equal to the transfer

buffer. Consider Figure 3.4 as an example. We arrive with vehicle 1 at 10:22 and with vehicle 2 at 10:27 at station C. The transfer buffer is set to 5 minutes. In fact we are able to transfer from vehicle 1 into vehicle 2 at C and reach D if we only keep the label concerning the earliest arrival time at C. However, assuming station D as the target station we get a shortest path as result where we have to transfer once. But the actual, optimal path to D with vehicle 2 has no transfer. Hence, we have to keep all labels where the number of transfers is the same, even if the time difference is greater than the transfer buffer.

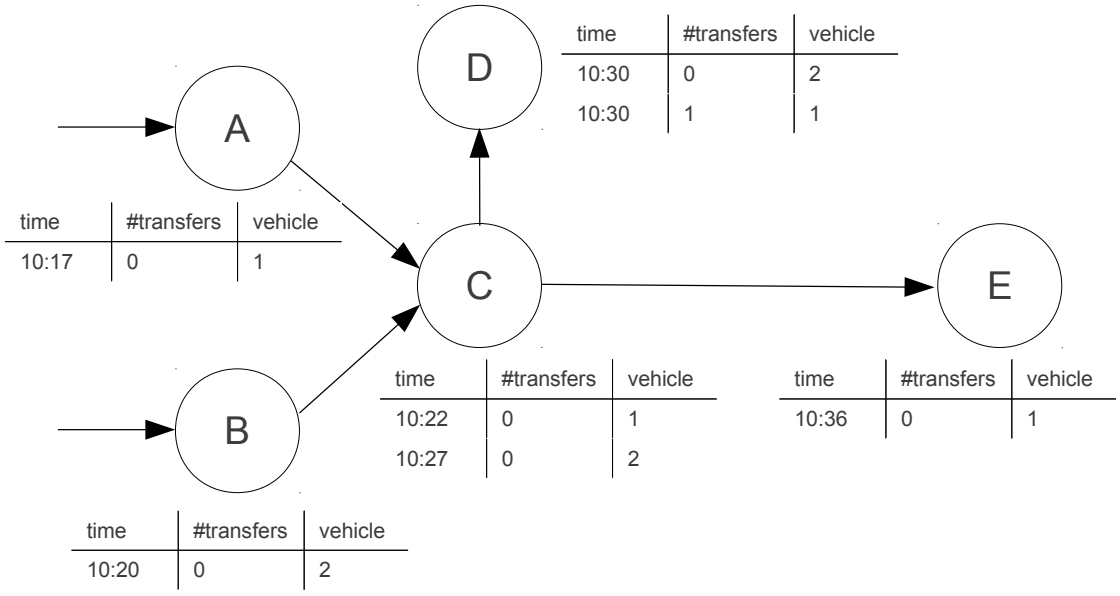

Figure 3.4: Time-dependent problem when using bi-criteria

4 Implementation

Based on the foundations in chapter 3 we explain the implementation of our system. First we give examples concerning the data we use. Then we explain how we construct the time-expanded and time-dependent graph including realistic features like transfer buffers and traffic days and combine each with the road network. As we do not only minimize the travel time when computing the shortest paths we focus on bi-criteria optimization concerning travel time together with the number of transfers. In addition, we also consider to minimize walking between and to stations. Further, we explain how we reduce the number of labels per station in the time-dependent model.

4.1 Data

4.1.1 GTFS Data

The General Transit Feed Specification (GTFS) [4], established by Google in 2005, defines a common format for exchanging public transit information. It allows transit agencies to publish their public transportation schedules and the associated geographic information and developers to write applications that use that data in an inter operable way.
The specification is divided into several feed files. As it is relatively complex in order to deal with all the special cases in a transit network, we will give a short overview and explain the basic concept. A station, respectively an individual location where vehicles pick up and drop off passengers is called a stop. A sequence of two or more stops that occurs at a specific time forms a trip. Trips that are displayed to riders as a single service are grouped to transit routes. For each trip of a vehicle the times when the vehicle arrives at and departs from individual stops are given. Further the days of the week when trips are available are specified.

4.1.1.1 Example GTFS Feed

The example GTFS feed (Figure 4.1) shows comma-delimited data samples for the files *stops.txt*, *stop_times.txt*, *trips.txt* and *calendar.txt* in a transit feed. These are the files we use in order to construct the transit graphs. The information when vehicles arrive at and leave stations is given in stop_time.txt. Having the stop id

stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url

FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,

BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,

BULLFROG,Bullfrog (Demo),,36.88108,-116.81797,,

STAGECOACH,Stagecoach Hotel & Casino (Demo),,36.915682,-116.751677,,

NADAV,North Ave / D Ave N (Demo),,36.914893,-116.76821,,

NANAA,North Ave / N A Ave (Demo),,36.914944,-116.761472,,

DADAN,Doing Ave / D Ave N (Demo),,36.909489,-116.768242,,

EMSI,E Main St / S Irving St (Demo),,36.905697,-116.76218,,

AMV,Amargosa Valley (Demo),,36.641496,-116.40094,,

Example stops.txt file

trip_id,arrival_time,departure_time,stop_id,stop_sequence,stop_headsign,pickup_type,drop_off_time,shape_dist_traveled

STBA,6:00:00,6:00:00,STAGECOACH,1,,,

STBA,6:20:00,6:20:00,BEATTY_AIRPORT,2,,,

CITY1,6:00:00,6:00:00,STAGECOACH,1,,,

CITY1,6:05:00,6:07:00,NANAA,2,,,

CITY1,6:12:00,6:14:00,NADAV,3,,,

CITY1,6:19:00,6:21:00,DADAN,4,,,

CITY1,6:26:00,6:28:00,EMSI,5,,,

CITY2,6:28:00,6:30:00,EMSI,1,,,

CITY2,6:35:00,6:37:00,DADAN,2,,,

CITY2,6:42:00,6:44:00,NADAV,3,,,

CITY2,6:49:00,6:51:00,NANAA,4,,,

CITY2,6:56:00,6:58:00,STAGECOACH,5,,,

Example stop_times.txt file

route_id,service_id,trip_id,trip_headsign,direction_id,block_id,shape_id

STBA,FULLW,STBA,Shuttle,,,

CITY,FULLW,CITY1,,0,,

CITY,FULLW,CITY2,,1,,

Example trips.txt file

service_id,monday,tuesday,wednesday,thursday,friday,saturday,sunday,start_date,end_date

FULLW,1,1,1,1,1,1,1,20070101,20101231

WE,0,0,0,0,0,1,1,20070101,20101231

Example calendar.txt file

Figure 4.1: Example GTFS Feed [3]

for each entry, we get information about the stations in stops.txt. For each trip id in trips.txt we get the service id in order to know which trips are available on which days (calendar.txt).

4.1.2 OSM Data

OpenStreetMap (OSM) [15] , founded in 2004, is a project whose aim is it to create a free and editable map of the whole world. The data, e.g. streets, railroad, rivers, houses etc., are collected by volunteers who typically use a GPS device. Therefore, OpenStreetMaps provides data that can be used and edited collectively all over the world. The raw data is provided in the XML format and can be downloaded for free.

4.1.3 OSM Examples

Nodes and *ways* [13], illustrated in Figure 4.2, are the basic components in OpenStreetMap from which everything else is defined. A node is a single geospatial point with latitude and longitude information. A way is an ordered list of between 2 and 2000 nodes. Ways can be used to represent linear features (vectors) or polygons (areas).
We search in the raw data provided in the XML format for the corresponding node and way tags in order to construct the road network. In Figure 4.2, for example, we have a one-way residential street, tagged as $highway = residential$ plus $name = Clipstone\ Street$ plus $oneway = yes$. The nodes defining the geometry of the way are enumerated in the correct order. They are indicated only by reference using their unique identifier. These nodes must have been already defined separately with their coordinates.

4.2 Constructing the Transit Network

As explained in subsection 4.1.1, each vehicle visits a sequence of stations at particular times. This sequence is called a trip. For each station concerning such a trip the timetable includes the arrival and departure times. Note that for the first and last station of a trip the arrival and departure times are equal. Consider, for example Table 4.1, where the timetable of three vehicles is given.

```
<node id="25496583" lat="51.5173639" lon="-0.140043" version="1" changeset="203496" user="80n" uid="1238" visible="true" timestamp="2007-01-28T11:40:26Z">
<tag k="highway" v="traffic_signals"/>
</node>
```

Example node

```
<way id="5090250" visible="true" timestamp="2009-01-19T19:07:25Z" version="8" changeset="816806" user="Blumpsy" uid="64226">
<nd ref="822403"/>
<nd ref="21533912"/>
<nd ref="821601"/>
<nd ref="21533910"/>
<nd ref="135791608"/>
<nd ref="333725784"/>
<nd ref="333725781"/>
<nd ref="333725774"/>
<nd ref="333725776"/>
<nd ref="823771"/>
<tag k="highway" v="residential"/>
<tag k="name" v="Clipstone Street"/>
<tag k="oneway" v="yes"/>
</way>
```

Example way

Figure 4.2: Example OSM Node [14] and OSM Way [16]

Table 4.1: Timetable of three vehicles

station	time
A	10:00 (dep)
B	10:28 (arr) 10:28 (dep)
C	10:55 (arr)

(a) vehicle 1

station	time
B	10:30 (dep)
D	11:10 (arr)

(b) vehicle 2

station	time
B	11:00 (dep)
C	11:20 (arr) 11:25 (dep)
D	11:40 (arr)

(c) vehicle 3

Table 4.2: Elementary connections of Table 4.1. According to the definition in section 3.1 we have the following elementary connections.

c_i	(	Z	S_1	S_2	t_d	t_a	)
c_1	(	1	A	B	10 : 00	10 : 28	)
c_2	(	1	B	C	10 : 28	10 : 55	)

(a) vehicle 1

c_i	(	Z	S_1	S_2	t_d	t_a	)
c_1	(	2	B	D	10 : 30	11 : 10	)

(b) vehicle 2

c_i	(	Z	S_1	S_2	t_d	t_a	)
c_1	(	3	B	C	11 : 00	11 : 20	)
c_2	(	3	C	D	11 : 25	11 : 40	)

(c) vehicle 3

Table 4.3: Example of a consistent connection. The table shows a consistent connection $P = (c_1, c_2, c_3)$ formed by three elementary connections of Table 4.2. There is one transfer from vehicle 1 to vehicle 3 at station B. As the transfer buffer is defined to be 5 minutes, a passenger has enough time to change vehicles. Hence, the consistency conditions in section 3.1 are not violated. Assume c_2 is replaced by vehicle 2 which departs from B at 10:30. As $2 < 5 = transfer(B)$ minutes, a passenger is not able to change vehicles. Hence replacing c_2 by vehicle 2 would not form a consistent connection.

c_i	(	Z	S_1	S_2	t_d	t_a	)
c_1	(	1	A	B	10 : 00	10 : 28	)
c_2	(	3	B	C	11 : 00	11 : 20	)
c_3	(	3	C	D	11 : 25	11 : 40	)

We illustrate the elementary connections of Table 4.1 in Table 4.2. Table 4.3 shows an example of a consistent connection and explains the difference to an inconsistent connection. In order to extend the definition of a consistent connection (see section 3.1), we define the departure and arrival times $t_d(c)$ and $t_a(c)$ of an elementary connection $c \in \mathcal{C}$ within a day to be integers in the interval $[0, 24 \cdot 3600 - 1]$ representing time in seconds after midnight. The departure times $dep_i(P)$ and arrival times $arr_i(P)$ include the departure respectively arrival day by counting time in seconds from the first day of the timetable. A time value t is defined as $t = a \cdot 24 \cdot 3600 + b$ with $a \in [0, 364]$ and $b \in [0, 24 \cdot 3600 - 1]$. According to this definition the actual time of a day is $t \ mod \ (24 \cdot 3600)$ and the actual day is $\lfloor t/(24 \cdot 3600) \rfloor$. Formally

a sequence of elementary connections $P = (c_1, \ldots, c_k)$ is consistent if the following conditions are satisfied:

1. c_i is valid on day $\lfloor dep_i(P)/(24 \cdot 3600) \rfloor$,
2. $S_2(c_i) = S_1(c_{i+1})$,
3. $dep_i(P) \equiv t_d(c_i)\ mod\ (24 \cdot 3600)$,
4. $arr_i(P) = dep_i(P) + length(c_i)$,
5. The minimum transfer durations are respected; either $Z(c_{i+1}) = Z(c_i)$ or $dep_{i+1}(P) - arr_i(P) \geq transfer(S_2(c_i))$

4.2.1 The Time-Expanded Network

Based on the GTFS data the time-expanded graph is constructed. For each arrival and departure event at a stop we construct an arrival and departure node with a direct edge from the arrival to the departure node. To model transfers, for each arrival node with time t we add a transfer node with time $t' = t + TB$, where TB is the transfer buffer, and add a direct edge from the arrival to the transfer node. Having created all transfer and departure nodes at each station and sorted them according to their time values in ascending order, for each transfer node we add an edge to the next transfer node in time and edges to all departure nodes without having a transfer node in-between. The cost of these edges is the time difference of the adjacent nodes. Figure 4.3 shows the time-expanded graph for the timetable in Table 4.1.

Traffic Days Integrating traffic days into the time-expanded model is done by only constructing arrival, departure and transfer nodes together with the corresponding edges for a traffic day if there is valid elementary connection between stations on that day. The times at nodes are stored as integers representing time in seconds after midnight. The times at these nodes are relative to the begin of the interval. For example, consider March 01, 2012, 00:00 as the begin of our timetable. A node in the time-expanded graph representing March 03, 2012 at 10:20 would hold the integer value $t = 2 \cdot 24 \cdot 3600 + 10 \cdot 3600 + 20 \cdot 60$.

4.2.2 The Time-Dependent Network

Based on the GTFS data the time-expanded graph is constructed. For each stop there is one node in the graph. Then according to the trips in the GTFS data the direct edges between stations are added. We store at an edge (A, B) for two stations A and B the departure time at A and the arrival time at B for a trip ℓ. We integrate the transfer buffer concerning station A into the edge (A, B). Thus, for each edge (A, B) we have a cost function $f_{(A,B)}(t, \ell)$ taking the arrival time t and

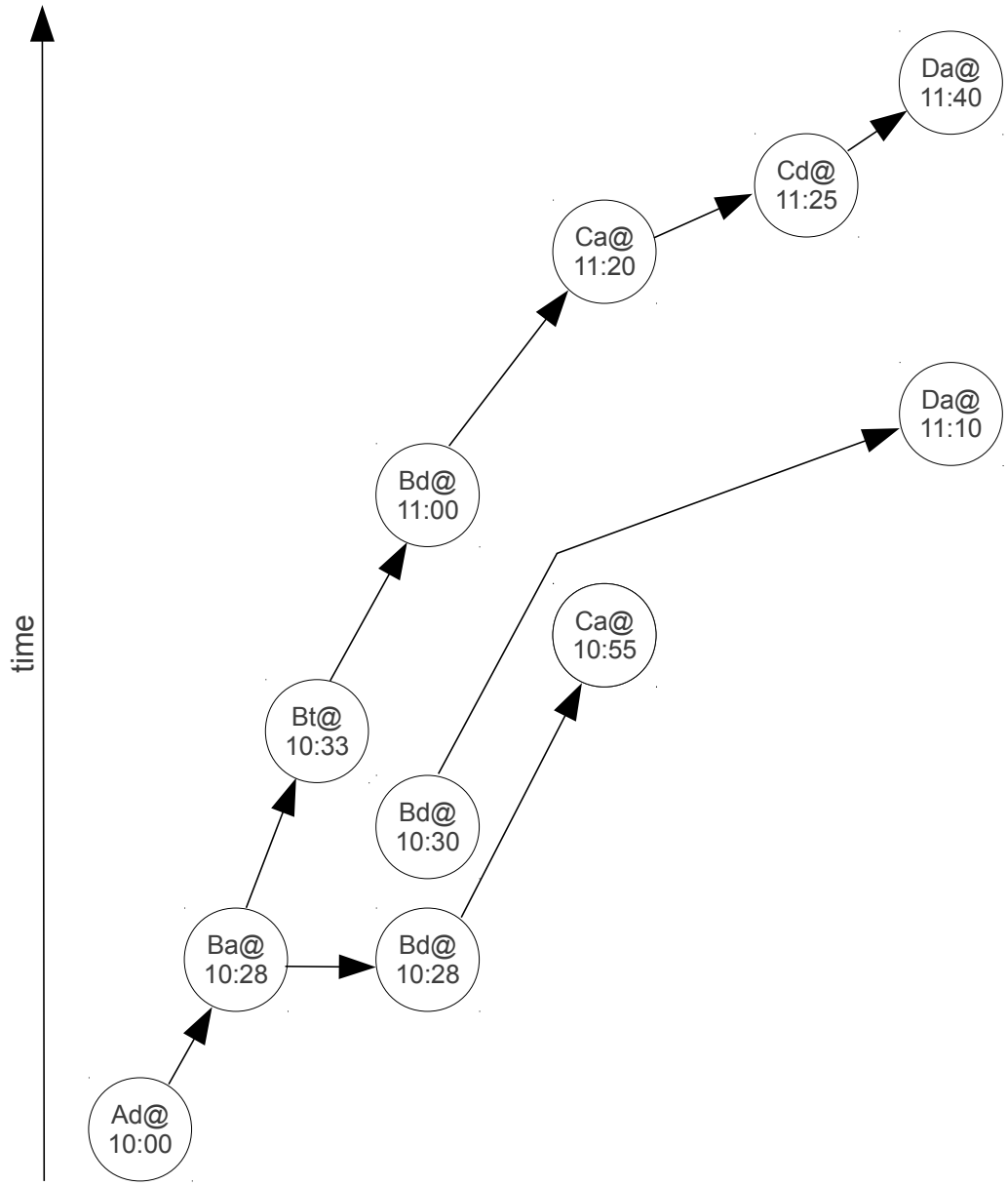

Figure 4.3: Time-expanded graph based on the timetable in Table 4.1. For each event there is a node and the nodes are arranged vertically concerning time. The type of a node is either denoted by an a for an arrival node, by a t for a transfer node or by a d for a departure node. For example, *Ad@10:00* is a departure node of station A at time 10:00. The cost of an edge is simply the difference between its endpoints. Note that the transfer buffer is set to 5 minutes. Therefore, a passenger is able to transfer from vehicle 1 to vehicle 3 at station B but not to vehicle 2.

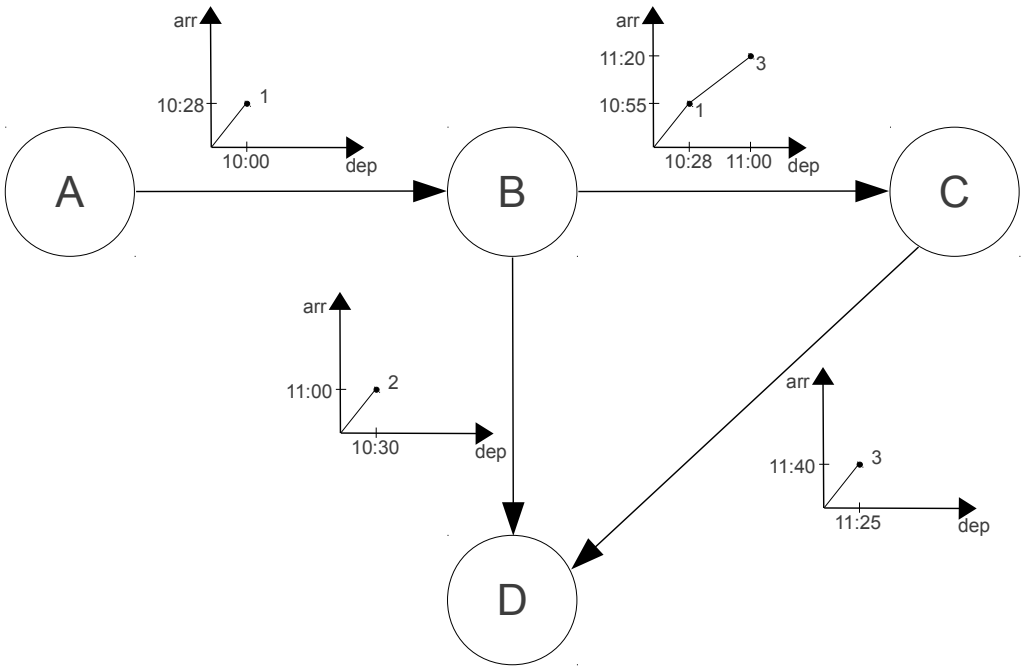

Figure 4.4: Time-dependent graph based on the timetable in Table 4.1. There is one node per station and each edge between two stations stores a cost function taking the arrival time and the trip id of the arriving vehicle. Consider, for example, the edge (B, C) where we arrive at 10:28 with vehicle 1. The cost function returns for arriving at station C the arrival times 10:55 (staying in vehicle 1) and 11:20 (changing to vehicle 3). Note that the transfer buffer is set to 5 minutes. As the transfer buffer is built into the cost functions, a passenger is able to transfer from vehicle 1 to vehicle 3 at station B but not to vehicle 2. Hence, arriving at B at 10:28 with vehicle 1 and changing vehicles requires a departure time at B which is greater than or equal to 10:33. This is the case for vehicle 3 but not for vehicle 2.

the trip id ℓ of the arriving vehicle at A. The cost function returns for all possible departure times at A the corresponding arrival times at B. Figure 4.4 shows the time-dependent graph for the timetable in Table 4.1.

Traffic Days Traffic days in the time-dependent network are handled in the cost functions. There is only a connection between two stations at a particular traffic day if there exists a valid elementary connection on that day. Similar to the time-expanded network, the times stored at edges are integers relative to the begin of the chosen interval.

4.3 Constructing the Road Network

Using the street information in the OSM data the road network is constructed as follows. Each junction represents a node in the graph. The edges are the road segments and are bidirectional. The cost of an edge is the time needed walking from one node to one of its neighbor nodes. Thus, the cost of an edge is the distance between the adjacent nodes divided by the walk velocity. The distance between two nodes is a path between two points on the earths surface which is a segment of a great circle [20]. The walk velocity is a constant which is set to 5 km/h.

4.4 Combining Road and Transit Network

In order to combine the road with the transit network we need to find the geographically closest OSM node in the road network to each station in the transit network. Then we have to insert new edges to link both networks accordingly and merge them to obtain the multi-modal network. As nodes in both networks store latitude and longitude values we use a k-dimensional tree [2] to find the geographically closest OSM node to each station. The idea of this data structure is to generalize a binary search tree to k dimensions. In our case, we have $k = 2$. In comparison to the naive approach where we scan the list of stations for each OSM node which takes linear time, we are able to find the nearest neighbor in average logarithmic time. The linking-operation of transit and road network is described in the following subsections. The merge-operation basically unifies the node and edge sets of multiple graphs.

4.4.1 Time-Expanded Network

Linking the road network with the time-expanded network is done by grouping all nodes of the time-expanded graph that logically belong to the same station. Then we introduce a new station arrival node holding no time value for each station and add a direct edge from the geographically closest OSM node of the road network to the station arrival node of a station. The cost of this edges is the distance between both nodes divided by the walk velocity. Further we add edges from the station arrival node to each departure node of a station and set the cost of each edge to the transfer buffer. Having added the station arrival nodes and corresponding edges which models entering a station we also have to model leaving a station. This is done by adding for each arrival node (not the station arrival node) of a station an edge to the geographically closest OSM node. The cost of this edge is the distance divided by the walk velocity. In Figure 4.5 we illustrate how road and time-expanded network are combined.

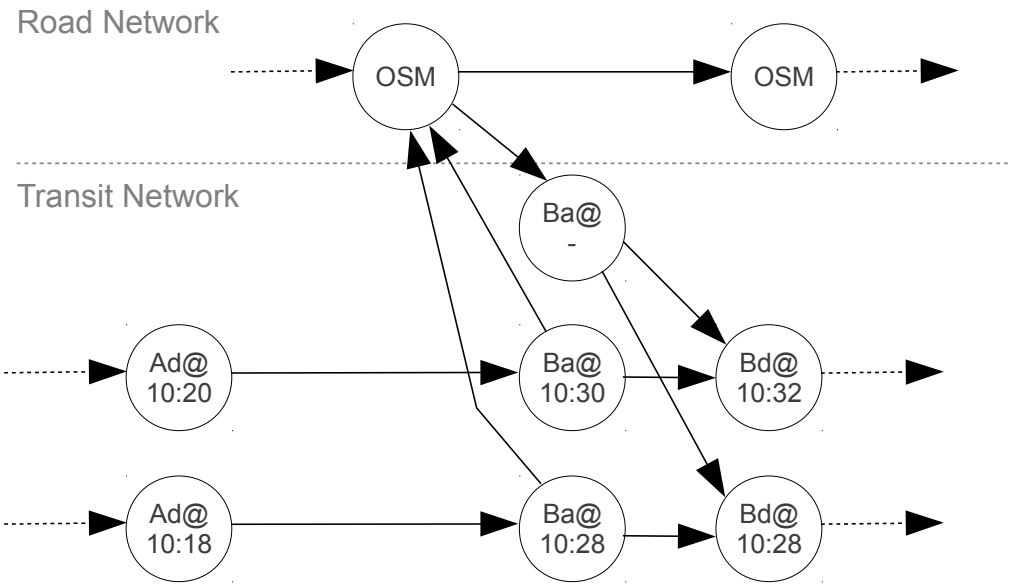

Figure 4.5: Combining road and time-expanded network. We have two arrival nodes at station B with each having edges to the closest OSM node. The station arrival node with one incoming edge from the closest OSM node and edges to each departure node of station B has no time value as the time when entering a station is determined while computing the shortest paths. Thus, in the Dijkstra computation we must ignore all edges between station arrival node and departure nodes where we arrive at a station and are not able to take a vehicle concerning time. The cost of an edge between station arrival node and departure node is the transfer buffer because entering a station and getting into an vehicle requires time, too. Thus, we use for getting into an vehicle or changing vehicles the same transfer buffer.

4.4.2 Time-Dependent Network

Linking the road with the time-dependent network is done by adding a bidirectional edge from the geographically closest OSM node in the road network to the corresponding node in the time-dependent network. The cost of this edge is the distance divided by the walk velocity. Note that we have cost functions for each edge in the combined time-dependent network. Thus, concerning edges in the road network and those coming from or to an OSM node, the cost function f returns for an edge (u, v) the arrival time at v which is the time we arrive at u plus the travel time we need to go from u to v.

4.5 Bi-criteria Optimization

In section 3.9 we explained the principles of multi-criteria where the goal is to find optimal connections under several criteria costs. In the implementation two criteria costs are used, namely the travel time and a so called penalty value explained in subsection 4.5.1. The goal is to compute all Pareto optimal solutions using a Multi-Label Dijkstra (see subsection 3.9.2). Hence, each node in the combined time-expanded and combined time-dependent network holds a set of 2-dimensional labels of the form (t, p) where t is the travel time and p the penalty value. The labels in the priority queue are processed according to some lexicographical ordering. Thus, a label (t, p) is processed before another label (t', p') if and only if $p < p' \vee (p = p' \wedge t < t')$.

4.5.1 Criteria Costs

As a second criteria cost, we use a penalty value that represents the number of transfers in the default setting. In the time-expanded network we increase the penalty value by 100 when we change vehicles which is the case when we relax an edge between an arrival and a transfer node. In the time-dependent network, for each arrival at a station S with trip id ℓ we increase the penalty value if for an arrival at the neighbor station S' with trip id ℓ' it holds that $\ell \neq \ell'$. Further, we increase in both networks the penalty value when we enter a station in order to take a vehicle and left another station beforehand over the corresponding link connecting road and transit network. Thus, we detect leaving and entering a station when the corresponding edges combining transit and road network are relaxed.
Optionally we want to minimize walking between and to stations when we walk for a minute or more. We remember for each label the walking time τ which is the time in seconds when relaxing edges in the road network or entering or leaving a station. Then for two labels (t, p) and (t', p'), where (t, p) is the predecessor of $(t'p')$ and we relax an edge in the road network or a linking edge, we increase the penalty value by $\lfloor \tau'/60 \rfloor - \lfloor \tau/60 \rfloor$.

4.5.2 Time-Expanded Network

In order to maintain the label set at each node, we need to define when a label is better, equal or incomparable to another label. Formally, for 2-dimensional labels of scalars of the form (t, p) we define the following:

1. $(t, p) \leq (t', p')$ if and only if $t \leq t'$ and $p \leq p'$,
2. $(t, p) < (t', p')$ if and only if $(t < t' \wedge p \leq p') \vee (t \leq t' \wedge p < p')$,
3. (t, p) (t', p') incomparable if neither$(t, p) \leq (t', p')$ nor $(t', p') \leq (t, p)$,

We add a label (t, p) to a set of labels when it either dominates some existing labels (t', p') such that $(t, p) < (t', p')$ and delete the dominated labels from the set or it

is incomparable to all other labels in the set. Otherwise if (t, p) is dominated by another label (t', p') in the set such that $(t', p') \leq (t, p)$, then we do not insert it.

4.5.3 Time-Dependent Network

Maintaining the label sets at nodes, we use label domination in the time-expanded sense if for two labels (t, p) and (t', p') with trip ids ℓ and ℓ' it holds that $\ell = \ell'$. However, when the trip ids are not equal we cannot use that definition any more. This is the case because we must keep labels for each node where the time difference of these labels is smaller than the transfer buffer for two different vehicles. Further, concerning bi-criteria, we must in addition keep all labels where the number of transfers is the same. For example, consider a label $(30, 2)$. We must keep all other labels $(t, 2)$ where $t \geq 30 + TB$ with TB being the transfer buffer. Hence, for 2-dimensional labels of scalars of the form (t, p) we define that:

1. $(t, p) \leq (t', p')$ if and only if $t \leq t' \wedge p \leq p'$ where $\ell = \ell'$,
2. $(t, p) < (t', p')$ if and only if $(t < t' \wedge p \leq p') \vee (t \leq t' \wedge p < p')$ where $\ell = \ell'$ or $t + TB \leq t' \wedge p < p'$
3. (t, p) (t', p') incomparable if $(t, p) \not\leq (t', p')$ and $(t', p') \not\leq (t, p)$ where $\ell = \ell'$ or $(t, p) \not< (t', p')$ and $(t', p') \not< (t, p)$

When minimizing walking we have to subtract the walking penalty portion from the entire penalty value as we are only interested to rule out labels where are able to transfer into another vehicle and do not loose any connections. Hence, when we minimize walking we define that:

1. $(t, p) \leq (t', p')$ if and only if $t \leq t' \wedge p \leq p'$ where $\ell = \ell'$,
2. $(t, p) < (t', p')$ if and only if $(t < t' \wedge p \leq p') \vee (t \leq t' \wedge p < p')$ where $\ell = \ell'$ or $t + TB \leq t' \wedge p - \lfloor \tau/60 \rfloor < p' - \lfloor \tau'/60 \rfloor \wedge \lfloor \tau/60 \rfloor \leq \lfloor \tau'/60 \rfloor$,
3. (t, p) (t', p') incomparable if $(t, p) \not\leq (t', p')$ and $(t', p') \not\leq (t, p)$ where $\ell = \ell'$ or $(t, p) \not< (t', p')$ and $(t', p') \not< (t, p)$

We add a label (t, p) to a set of labels when it dominates some existing labels (t', p') such that $(t, p) < (t', p')$ and delete the dominated labels from the set. We also add (t, p) if it is incomparable to all other labels in the set. Otherwise if (t, p) is dominated by another label (t', p') in the set such that $(t', p') \leq (t, p)$ or $(t', p') < (t, p)$, then we do not insert it.

4.5.4 Ignoring Labels using Multi Label Dijkstra

In the ordinary Dijkstra computation we can stop when we reach the target node. Using the multi label Dijkstra we cannot stop when we reach the target node as we want to find all optimal solutions. Note that start and target nodes are OSM nodes, thus concerning the time-expanded model we do not have the problem of checking

whether the target node belongs to the target station.
In both models we ignore labels that do not lead to an optimal path any more. Hence for each label (t, p) popped from the priority queue we check whether we have an label (t', p') at the target node such that $t' \leq t \wedge p' \leq p$. If this is the case we ignore (t, p), i.e. we do not relax it.

4.6 Reducing Labels per Station in the Time-Dependent Model

The problem in the time-dependent model is that we have too many labels per station which makes the calculation of the shortest path inefficient. But we are able to reduce the number of labels per station. As different trips may follow at some point the same sequence of stations, we can reduce the number of labels per stations as follows. For each trip pair where both trips end at the same station, we remember how many stations both trips have in common up to the final station. Then for two labels (t, p) and (t', p') with trip ℓ and ℓ' at a station such that both trips follow the same sequence of stations to the final station we use the definition of label domination in the time-expanded sense (see subsection 4.5.2). The example in Figure 4.6 illustrates this.

In order to store trip pairs with the corresponding station sequence we need to do a precomputation and store the information in a hash map. Thus, we have for the hash map h the mapping $h : (\ell, \ell') \to (arr_\ell(S), arr_{\ell'}(S))$ where ℓ, ℓ' are trip ids and $arr_\ell(S)$, $arr_{\ell'}(S)$ are the times when we arrive at station S over the corresponding trip. In fact we do not store the sequence of common stations for each pair of trips but for the first station S in that sequence the times when we arrive at that station over the corresponding trips. Now, if we have two labels (t, p) and (t', p') with the trips ℓ and ℓ' at a station S we make a look up in the hash map. Provided there is an entry we compare the times of the labels with the stored times in the hash map such that $t \geq arr_\ell(S) \wedge t' \geq arr'_{\ell'}(S)$. If this holds we use the definition for label domination in the time-expanded sense to compare the labels. In case $\ell = \ell'$ we do not have an entry in the hash map as we are able to apply label domination in the time-expanded sense directly. Note that the relation of the times between two trips is important, i.e. if we had, for example, in Figure 4.6 an arrival time of $10 : 55$ for trip 1 at station E we cannot rule out label $(10 : 20, 2)$ at C because we loose a better connection. Hence, we cannot make an entry in the hash map for the trips and the times at station C but make the entry for station E.

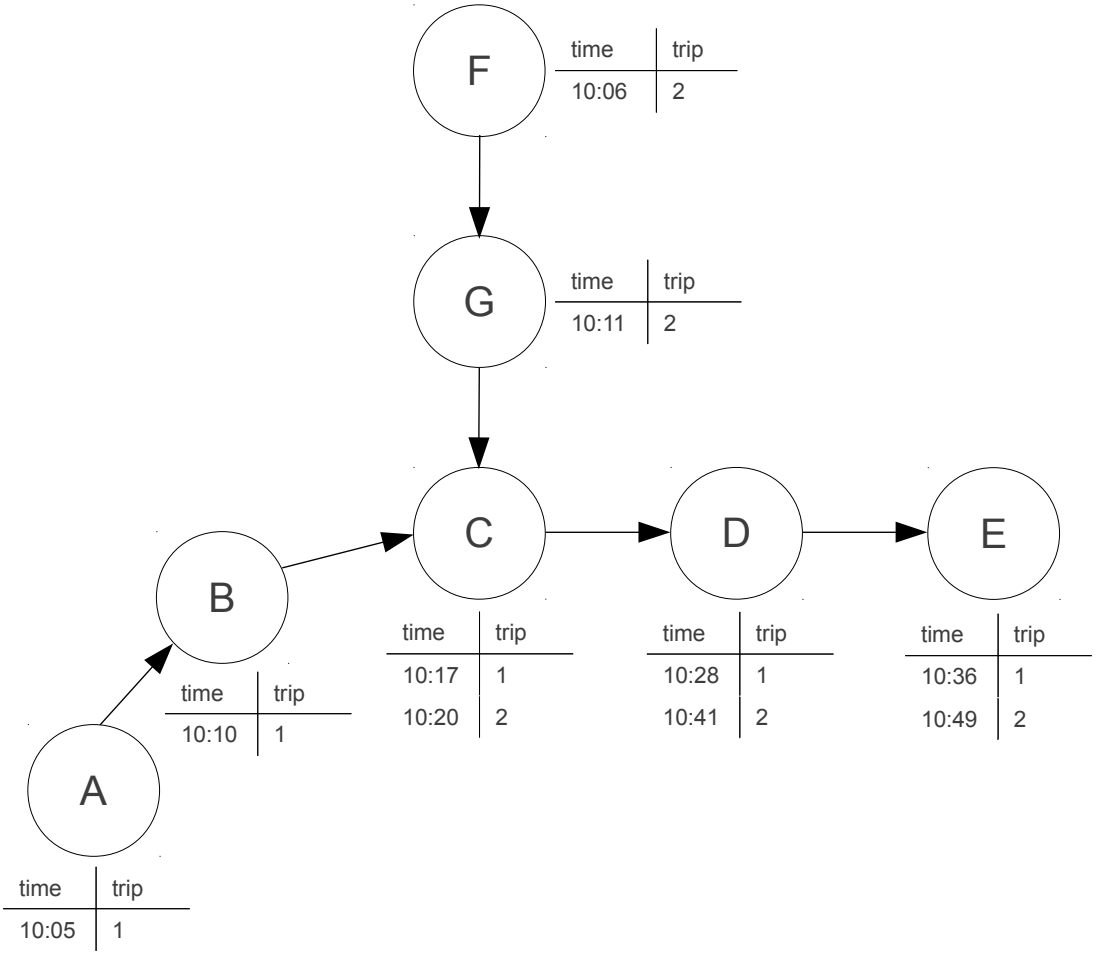

Figure 4.6: Two different trips following at some point the same sequence of stations. Trip 1 follows the stations A,B,C,D,E whereas trip 2 follows F,G,C,D,E. Both trips have station E as the final station. The times and trips at each node shows at what time over which trip we arrive at the corresponding station. Now, assume we have the label $(10:17, 2)$, where the second value is the number of transfers, for trip 1 at station C. Then we create another label $(10:20, 2)$ for trip 2 at station C. As we know that from station C onwards both trips follow the same sequence of stations to the final station E, we can apply the definition of label domination in the time-expanded sense and rule out label $(10:20, 2)$.

5 Evaluation

In this chapter we prove, in the first part, that the sequence of labels settled in the time-dependent computation is a subsequence of those labels settled in the time-expanded computation, provide the labels are processed in the same order by the priority queue. In the second part we compare both models concerning graph size and performance by calculating 1000 random paths using single criterion and bi-criteria optimization.

5.1 Label Correspondence

The computation of the shortest paths in the time-expanded and time-dependent model is characterized by a sequence of labels processed by the priority queue according to the lexicographical ordering $p < p' \vee (p = p' \wedge t < t')$ for two labels (t, p) and (t', p'). In the following we prove that the label sequence of the time-dependent model is a subsequence of the label sequence of the time-expanded model.

Definition 1: Equality of labels. *Two labels of the form (t, p) and (t', p') with trips ℓ and ℓ' are equal if and only if $t = t' \wedge p = p' \wedge \ell = \ell'$ and both labels belong to the same station. We refer to the sequence of labels in the time-expanded model (time-dependent model) as seqExp (seqDep). As in seqExp we have labels belonging to arrival, transfer and departure nodes we refer to these labels as arrival, transfer and departure labels. Labels belonging to OSM nodes are referred to as OSM labels. In the following we denote the transfer buffer by TB.*

Theorem 1. *The sequence of labels seqDep processed in the time-dependent computation is a subsequence of the sequence of labels seqExp processed in the time-expanded computation using only the number of transfer as penalty value.*

Proof. In *seqDep* and *seqExp* we assume an ordering of labels such that for two labels (t, p) and (t', p') we have that $(t, p) < (t', p')$ if and only if $p < p' \vee (p = p' \wedge t < t')$. We want to prove that $seqDep \subseteq seqExp$. We show the following by induction over i and j. In iteration i concerning the computation of the time-dependent model we settle a label $(t_i, p_i)_{Dep} \notin seqDep'$. Then we settle an arrival

or OSM label $(t_j, p_j)_{Exp} \notin seqExp'$ equal to $(t_i, p_i)_{Dep}$ in iteration j of the time-expanded model such that $seqDep' \subseteq seqExp'$ holds, where $seqDep' \subseteq seqDep$ and $seqExp' \subseteq seqExp$. We assume that for all arrival and OSM labels contained in $seqExp'$ we have an equal label contained in $seqDep'$. Thus, in our proof we use for the time-expanded and time-dependent model label domination in the time-expanded sense (see subsection 4.5.2), which means that a label $(t, p)_{Exp} \in seqExp'$, respectively $(t, p)_{Dep} \in seqDep'$, dominates another label $(t', p')_{Exp} \in seqExp'$, respectively $(t', p')_{Dep} \in seqDep'$, at a node, respectively station, if and only if $(t < t' \wedge p \leq p') \vee (t \leq t' \wedge p < p')$ where $\ell = \ell'$.

Concerning the base case, for $i = 1$ and $j = 1$ we settle in both models the start label $(t_1, p_1)_{Dep}$ and $(t_1, p_1)_{Exp}$ which is equal for both models. Hence, $seqDep' \subseteq seqExp'$ holds.

We assume that in iteration i and j the claim holds. Then in iteration $i + 1$ we settle label $(t_{i+1}, p_{i+1})_{Dep} \notin seqDep'$. We show that the first settled arrival or OSM label $(t_{j+n}, p_{j+n})_{Exp} \notin seqExp'$, where $n \geq 1$, in the time-expanded model is equal to $(t_{i+1}, p_{i+1})_{Dep}$. In the following we prove that (1) the label $(t_{j+n}, p_{j+n})_{Exp}$ is settled, i.e it is not dominated by another arrival or OSM label. Then (2) we show that $(t_{j+n}, p_{j+n})_{Exp} \notin seqExp'$ is in fact the first settled arrival or OSM label equal to $(t_{i+1}, p_{i+1})_{Dep}$, i.e. there exists no other arrival or OSM label $(t_{j+l}, p_{j+l})_{Exp} \notin seqExp'$ with $(t_{j+l}, p_{j+l})_{Exp} < (t_{j+n}, p_{j+n})_{Exp}$ and $l < m$ and $(t_{j+l}, p_{j+l})_{Exp}$ is equal to $(t_{i+1}, p_{i+1})_{Dep}$.

We prove part 1. In the following we rename label $(t_{i+1}, p_{i+1})_{Dep}$ to $(t, p)_{Dep}$ and label $(t_{j+n}, p_{j+n})_{Exp}$ to $(t, p)_{Exp}$. The label $(t', p')_{Dep}$ is the predecessor label of label $(t, p)_{Dep}$. As label $(t', p')_{Dep}$ is contained in $seqDep'$, we have by assumption an equal label $(t', p')_{Exp} \in seqExp'$. We distinguish 5 cases: (1.1) $(t, p)_{Dep}$ and $(t', p')_{Dep}$ are OSM labels at OSM nodes n and n' where n' has an edge to n, (1.2) $(t, p)_{Dep}$ is a label at station S and $(t', p')_{Dep}$ is a OSM label at OSM node n' where n' has an edge to S, (1.3) $(t, p)_{Dep}$ is a OSM label at OSM node n and $(t', p')_{Dep}$ is a label at station S' where S' has an edge to n, (1.4) $(t, p)_{Dep}$ and $(t', p')_{Dep}$ are labels at stations S and S' with $\ell = \ell'$ where S' has an edge to S, (1.5) $(t, p)_{Dep}$ and $(t', p')_{Dep}$ are labels at stations S and S' with $\ell \neq \ell'$ where S' has an edge to S .
1.1) We know that we create label $(t, p)_{Exp}$ equal to $(t, p)_{Dep}$ as the time-dependent and time-expanded model use the same information concerning the road network. Suppose that label $(t, p)_{Exp}$ is dominated by another label $(t'', p'')_{Exp} \notin seqExp'$ with $(t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)$ at OSM node n. Then there must be an OSM label $(t''', p''')_{Exp} \in seqExp'$ which is the predecessor of $(t'', p'')_{Exp}$ and not equal to $(t', p')_{Exp}$. By assumption we also have the label $(t''', p''')_{Dep} \in seqDep'$ equal to $(t''', p''')_{Exp}$. But then we have label $(t'', p'')_{Dep} \notin seqDep'$ at n, where $(t''', p''')_{Dep}$ is the predecessor of $(t'', p'')_{Dep}$, and $(t'', p'')_{Dep}$ dominates the label $(t, p)_{Dep}$ with $(t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)$. Hence, if we settle $(t, p)_{Dep}$ the label $(t'', p'')_{Exp}$ dominating $(t, p)_{Exp}$ cannot exist.

1.2) We know that we create label $(t,p)_{Exp}$ equal to $(t,p)_{Dep}$ as each station in the time-dependent and time-expanded model is linked to the geographically closest OSM node (see section 4.4) which is equal in both models concerning latitude and longitude information. Concerning the time-expanded model we introduced a new arrival node u for entering a station (see subsection 4.4.1). Thus, we create $(t,p)_{Exp}$ at u of station S as we have an edge from the geographically closest OSM node n' to u. Suppose that label $(t,p)_{Exp}$ is dominated by another label $(t'',p'')_{Exp} \notin seqExp'$ with $(t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)$ at u. Then there must be a label $(t''',p''')_{Exp} \in seqExp'$ with $(t''' < t' \wedge p''' \leq p') \vee (t''' \leq t' \wedge p''' < p')$, which is the predecessor of $(t'',p'')_{Exp}$, at OSM node n'. By assumption we also have the label $(t''',p''')_{Dep} \in seqDep'$ equal to $(t''',p''')_{Exp}$ at n'. But then we have $(t'',p'')_{Dep} \notin seqDep'$ at station S, where $(t''',p''')_{Dep}$ is the predecessor of $(t'',p'')_{Dep}$, and $(t'',p'')_{Dep}$ dominates label $(t,p)_{Dep}$ with $((t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)) \wedge \ell'' = \ell$. Hence, if we settle $(t,p)_{Dep}$ the label $(t'',p'')_{Exp}$ dominating $(t,p)_{Exp}$ cannot exist.
1.3) We know for the same reason as in 1.2 that we create label $(t,p)_{Exp}$ equal to $(t,p)_{Dep}$. Suppose that label $(t,p)_{Exp}$ is dominated by another label $(t'',p'')_{Exp} \notin seqExp'$ with $(t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)$ at OSM node n. Then there must be an arrival label $(t''',p''')_{Exp} \in seqExp'$ with $(t''' < t' \wedge p''' \leq p') \vee (t''' \leq t' \wedge p''' < p')$, which is the predecessor of $(t'',p'')_{Exp}$. By assumption we also have the label $(t''',p''')_{Dep} \in seqDep'$ equal to $(t''',p''')_{Exp}$. But then we have label $(t'',p'')_{Dep} \notin seqDep'$ at OSM node n, where $(t''',p''')_{Dep}$ is the predecessor of $(t'',p'')_{Dep}$, and $(t'',p'')_{Dep}$ dominates the label $(t,p)_{Dep}$ with $((t'' < t \wedge p'' \leq p) \vee (t'' \leq t \wedge p'' < p)) \wedge \ell'' = \ell$. Hence, if we settle $(t,p)_{Dep}$ the label $(t'',p'')_{Exp}$ dominating $(t,p)_{Exp}$ cannot exist.
1.4) We know that we create label $(t,p)_{Exp}$ equal to $(t,p)_{Dep}$ as the time-dependent and time-expanded model use the same timetable information. We suppose that $(t,p)_{Exp}$ is dominated by another label $(t'',p'')_{Exp} \notin seqExp'$ with $t'' = t \wedge p'' < p \wedge \ell'' = \ell$ at station S. Then there must be an arrival label $(t''',p''')_{Exp} \in seqExp'$ with $t''' + TB \leq t' \wedge p''' < p' \wedge \ell''' \neq \ell' \wedge \ell''' \neq \ell''$ at station S'. Note that the labels $(t''',p''')_{Exp}$ and $(t',p')_{Exp}$ belong to different arrival nodes. By assumption we have the label $(t''',p''')_{Dep} \in seqDep'$ equal to $(t''',p''')_{Exp}$ at S'. But when we create $(t'',p'')_{Dep}$, where $(t''',p''')_{Dep}$ is the predecessor, and $(t'',p'')_{Dep}$ dominates label $(t,p)_{Dep}$ such that $t'' = t \wedge p'' < p \wedge \ell'' = \ell$, then we do not have $(t,p)_{Dep}$. Hence, if we settle $(t,p)_{Dep}$ the label $(t'',p'')_{Exp}$ dominating $(t,p)_{Exp}$ cannot exist.
1.5) We know for the same reason as in 1.4 that we create label $(t,p)_{Exp}$ equal to $(t,p)_{Dep}$. We suppose that $(t,p)_{Exp}$ is dominated by another label $(t'',p'')_{Exp} \notin seqExp'$ with $t'' = t \wedge p'' < p \wedge \ell'' = \ell$. Then there must be an arrival label $(t''',p''')_{Exp} \in seqExp'$ where it either (1.5.1) holds that $t''' \leq t' \wedge p''' < p' \wedge \ell''' \neq \ell' \wedge \ell''' \neq \ell''$ or (1.5.2) $t''' \geq t' + TB \wedge p''' \leq p' \wedge \ell''' \neq \ell' \wedge \ell''' = \ell''$ at station S'. Note that the labels $(t''',p''')_{Exp}$ and $(t',p')_{Exp}$ belong to different arrival nodes. For 1.5.1 and 1.5.2 we know by assumption that we have the label $(t''',p''')_{Dep} \in seqDep'$ equal to $(t''',p''')_{Exp}$ at S'. But when we create $(t'',p'')_{Dep}$, where $(t''',p''')_{Dep}$ is the predecessor, and $(t'',p'')_{Dep}$ dominates label $(t,p)_{Dep}$ such that $t'' = t \wedge p'' < p \wedge \ell'' = \ell$

, then we do not have $(t,p)_{Dep}$. Hence, if we settle $(t,p)_{Dep}$ the label $(t'',p'')_{Exp}$ dominating $(t,p)_{Exp}$ cannot exist.

We prove part 2. Suppose we settle an arrival or OSM label $(t_{j+l},p_{j+l})_{Exp} \notin seqExp'$ with $(t_{j+l},p_{j+l})_{Exp} < (t_{j+n},p_{j+n})_{Exp}$ and $l < n$ which is not equal to $(t_{i+1},p_{i+1})_{Dep}$. Then we have an arrival or OSM label $(t_m,p_m)_{Exp} \in seqExp'$ with $m \leq j$ and $(t_m,p_m)_{Exp} < (t_{j+l},p_{j+l})_{Exp}$. As we defined $(t_{j+l},p_{j+l})_{Exp}$ to be the first label not contained in $seqExp'$ and $(t_m,p_m)_{Exp} < (t_{j+l},p_{j+l})_{Exp}$ holds, $(t_m,p_m)_{Exp}$ is contained in $seqExp'$.
By assumption we have the label $(t_k,p_k)_{Dep} \in seqDep'$ with $k \leq i$ which is equal to $(t_m,p_m)_{Exp}$. But then we create a label $(t_l,p_l)_{Dep} \notin seqDep' < (t_{i+1},p_{i+1})_{Dep}$ with $l > i \wedge l < i+1$. As in iteration $i+1$ the label $(t_l,p_l)_{Dep} \notin seqDep'$ cannot exist, label $(t_{j+l},p_{j+l})_{Exp}$ does not exist either. Hence, in iteration $j+n$ the first settled arrival or OSM label $(t_{j+n},p_{j+n})_{Exp} \notin seqExp'$ is equal to $(t_{i+1},p_{i+1})_{Dep} \notin seqDep'$. □

5.2 Experiments

We compare the time-expanded with the time-dependent approach concerning graph size and performance. The main goal is to compare the performance of both approaches using single criterion and bi-criteria optimization.

5.2.1 Experimental Setup

All of our code was written in *C++* and compiled with *GNU C++ compiler version 4.4.3.* We ran our experiments on a Intel Xeon X5560 2.8GHz machine with 16 processors, 40 GB of main memory, and running Linux. The data set [8] we used concerning the public transportation network comprises the New York City Subway, New York City Bus (Bronx, Brooklyn, Manhattan, Queens, Staten Island), Long Island Bus and the Metro-North Railroad. Concerning the road network we used a data set [6] comprising the state New York. We constructed the time-expanded and time-dependent graph each for one week in the date interval April 09th, 2012 to April 15th, 2012.

5.2.2 Comparing the Graph Size

Table 5.1 shows the number of nodes and edges and the memory consumption for the time-expanded, denoted Exp, and for the time-dependent approach, denoted Dep. We list the number of nodes concerning road and transit network as well as for the combined graph. In addition, we give the number of hash map entries as well as its memory consumption. We use the hash map to reduce the number of labels per station in the time-dependent model (section 4.6). Concerning the road

network we have about 4.5 million nodes and about 16.7 thousand stations in the transit network. As we built the graphs for one week (7 days), the time-expanded graph is about 7 times larger than if we would have built it for one day. It needs about 10.7 GB of memory. In contrast the time-dependent graph needs only 1.6 GB of memory.

Table 5.1: Number of nodes and edges of time-expanded and time-dependent graph

	#nodes			#edges	space	hash map	
	road	transit	combined			entries	space
Exp	4.5 M	43.8 M	48.3 M	112.4 M	10 670 MB	-	-
Dep	4.5 M	16.7 K	4.5 M	9.4 M	1 643 MB	49.0 M	1 567 MB

5.2.3 Comparing Performance

In the following we compare the performance of the time-expanded and time-dependent approach. In each table we give the run time, the number of label (label ops) and relax operations (relax ops) together with the time, the number of settled labels and in the time-dependent case we also give the average number of labels per station. The label operations are the operations we need to maintain the label sets at nodes. In fact, we count how many operations we have where we add a label to a label set or discard a label because another label in the set dominates it. The relax operations are those where we relax labels in order to create new labels at the adjacent node. We carried out each experiment on 1000 random paths where the transfer buffer was set to 5 minutes. Among these 1000 random paths we computed for each approach the average (av), median (50), 90%-ile (90), 99%-ile (99) and the maximum (max) concerning run time, label and relax operations and time, settled labels and average number of labels per station.

Single Criterion Table 5.2 shows the results of computing 1000 random paths using single criterion optimization. We minimized the travel time in order to find the fastest connection to get from A to B. Concerning the average run time, the time-dependent approach, denoted as Dep(av), is about four times faster than the time-expanded approach, denoted as Exp(av). This is due to the reduced search space as those labels are ignored which do not provide a better path in terms of travel time (see subsection 3.6.3).

Bi-criteria Table 5.3 shows the results of computing 1000 random paths using bi-criteria optimization. We computed all Pareto optimal solutions using travel time as first and the number of transfers as second criteria cost. Comparing to the single criterion optimization the run time increases as in the time-expanded (Exp) and time-dependent model (Dep) we have label sets at nodes (see section 4.5). Thus, in

Table 5.2: Benchmark concerning single criteria optimization

	time	label ops		relax ops		settled labels	labels / station
		#ops	time	#ops	time		
Exp(av)	23.9 s	14.4 M	4.7 s	14.4 M	4.1 s	6.9 M	-
Dep(av)	5.4 s	8.7 M	0.8 s	8.7 M	2.6 s	0.9 M	1
Exp(50)	8.0 s	4.3 M	1.6 s	4.3 M	1.4 s	2.0 M	-
Dep(50)	3.1 s	2.9 M	0.3 s	2.9 M	1.6 s	0.4 M	1
Exp(90)	19.4 s	11.4 M	3.8 s	11.4 M	3.3 s	5.5 M	-
Dep(90)	4.8 s	7.3 M	0.7 s	7.3 M	2.4 s	0.7 M	1
Exp(99)	22.7 s	13.6 M	4.4 s	13.6 M	3.9 s	6.5 M	-
Dep(99)	5.3 s	8.5 M	0.8 s	8.5 M	2.6 s	0.9 M	1
Exp(max)	156.4 s	110.7 M	35.4 s	110.7 M	29.5	47.6 M	-
Dep(max)	19.6 s	28.4 M	4.7 s	28.4 M	8.9 s	4.4 M	2

the average case the number of settled labels for Exp(av) increases from 6.9 million to 20.7 million labels whereas for Dep(av) 2 million labels are settled compared to 0.9 million. As we have concerning Dep(av) only 3 labels per station in average, one label operation takes about 0.4 μs which is the same time we have for Exp(av) for one label operation. Thus, the speed-up factor of Dep(av) compared to Exp(av) concerning run time is 3.7. However, the performance gets worse when we minimize walking in addition. Table 5.4 shows the results of computing 1000 random paths using bi-criteria optimization. We computed all Pareto optimal solutions using travel time as first and the number of transfers together with the walking penalty as second criteria cost. Compared to Table 5.3 the number of labels per station in average increased for Dep(av) from 3 to 13. Thus, we have for Dep(av) about 0.6 μs per label operation whereas for Exp(av) we still have 0.4μs. Considering Dep(max) and Exp(max) we have concerning time per label operation 1.3 μs compared to 0.5 μs. As maintaining the label sets for Dep is less efficient when the number of labels per station increases, the speed-up factor of Dep(av) compared to Exp(av) concerning run time is only 1.5. Concerning Dep(max) and Exp(max), the time-dependent approach is slower as we have 32 labels per station in average.

Table 5.3: Benchmark concerning bi-criteria optimization (travel time and number of transfers)

	time	label ops		relax ops		settled labels	labels / station
		#ops	time	#ops	time		
Exp(av)	62.3 s	32.8 M	13.2 s	32.8 M	10.3 s	20.7 M	-
Dep(av)	16.7 s	24.2 M	8.7 s	24.2 M	3.7 s	2.0 M	3
Exp(50)	15.8 s	7.9 M	3.4 s	7.9 M	3.0 s	4.9 M	-
Dep(50)	5.9 s	6.0 M	2.2 s	6.0 M	1.5 s	0.7 M	2
Exp(90)	50.2 s	26.0 M	10.7 s	26.0 M	8.2 s	16.8 M	-
Dep(90)	14.0 s	19.5 M	7.2 s	19.5 M	3.1 s	1.6 M	3
Exp(99)	60.5 s	31.7 M	12.8 s	31.7 M	10.0 s	20.2 M	-
Dep(99)	16.2 s	23.6 M	8.5 s	23.6 M	3.6 s	1.9 M	3
Exp(max)	311.0 s	202.4 M	73.4 s	202.4 M	59.0 s	93.8 M	-
Dep(max)	85.9 s	114.7 M	33.8 s	114.7 M	24.0	21.3 M	9

Table 5.4: Benchmark concerning bi-criteria optimization (travel time and number of transfers together with minimizing walking)

	time	label ops		relax ops		settled labels	labels / station
		#ops	time	#ops	time		
Exp(av)	89.2 s	64.1 M	27.3 s	64.1 M	16.2 s	33.8 M	-
Dep(av)	58.6 s	73.1 M	42.2 s	73.1 M	8.4 s	2.4 M	13
Exp(50)	18.8 s	12.3 M	5.2 s	12.3 M	4.0 s	7.2 M	-
Dep(50)	9.1 s	11.4 M	3.9 s	11.4 M	2.5 s	0.9 M	7
Exp(90)	67.7 s	48.9 M	20.2 s	48.9 M	12.6 s	26.3 M	-
Dep(90)	40.5 s	55.4 M	27.4 s	55.4 M	6.7	2.0 M	12
Exp(99)	85.8 s	61.9 M	26.1 s	61.9 M	15.7 s	32.7 M	-
Dep(99)	55.1 s	70.6 M	39.3 s	70.6 M	8.1 s	2.3 M	13
Exp(max)	527.1 s	341.1 M	185.4 s	341.1 M	82.2 s	168.7 M	-
Dep(max)	619.1 s	393.0 M	492.9 s	393.0 M	102.0 s	18.2 M	32

6 Discussion

This chapter concludes the work we have done as well as the results we have found in this thesis. In addition, we provide future work in order to extend our system concerning more realistic features and to reduce the query time.

6.1 Conclusion

We presented our system that solves shortest path problems on combined road and public transportation networks using the time-expanded and the time-dependent approach. We evaluated both approaches for single as well as bi-criteria problems including realistic features like traffic days and transfer buffers. Concerning bi-criteria, we used the travel time as the first and the number of transfers as the second criteria cost. The experiments show that when using bi-criteria optimization the time-dependent approach is about 3.7 times faster in average than the time-expanded approach, as maintaining the label sets at stations can be done efficiently when having relatively few labels per station in average. However, when minimizing walking between stations in addition, the gap is smaller as we get more optimal and incomparable solutions when computing the shortest paths. Thus, in the time-dependent approach the average number of labels per station increases and the speed-up factor is only 1.5.
Concerning the time-dependent graph we have one node per station. Hence, the time-dependent graph is much smaller than the time-expanded graph. Especially when building the graphs for several days, the size of the time-expanded graph grows by a factor of the number of days used.
We showed that the sequence of processed labels in the time-dependent approach which characterizes the computation of the shortest paths is a subsequence of the labels processed in the time-expanded approach, provided we use the same label order in the priority queue of both Dijkstra computations. In fact, as the experiments show, the time-dependent approach has a reduced search space compared to the time-expanded approach. On the one hand, this is the case as the time-dependent approach realizes waiting and departing at stations in the cost functions while the time-expanded approach relaxes labels at transfer and departure nodes. On the other hand we reduce the number of labels per station by discarding labels when we are able to transfer into another vehicle and do not loose a connection. Further, we use the precomputed hash map to remove connections that follow the same sequence of stations. This makes maintaining the label sets in the time-dependent approach

more difficult but reduces the number of labels per station. Concerning the time-expanded approach, it is easier to integrate the number of transfers and the walking penalty as second criteria cost and maintain the label sets at nodes accordingly.

6.2 Future Work

In the following we provide future work in order to extend the system by including more realistic features like exceptions and additional transfer rules for operating vehicles. Further we show how to reduce the size of the time dependent graph and suggest the transfer pattern algorithm to reduce the query time.

Exceptions Exceptions concerning GTFS data are, for example, holidays where particular vehicles do not operate. So far, we only considered vehicles operating for a special day of the week ignoring exceptions. The GTFS specification provides this information in the optional feed file *calendar_dates.txt* (see [4] for details) by giving the date when an exception for a vehicle takes place. That means that it is possible to delete or add an operating vehicle from or to the timetable for a particular date. In order to integrate exceptions into our implementation we have to do the following. For each trip of a vehicle together with the service id (see subsubsection 4.1.1.1), we know at which days of the week the trip is available. The service id in combination with a date, concerning exceptions, is then used to state whether a trip is added or removed for the specified date provided there is an entry in *calendar_dates.txt.* As before, we only have a connection between stations if a trip is available for a date.

Additional Transfer Rules In our implementation we used a constant transfer buffer for all stations which was set to 5 minutes. The GTFS specification provides additional rules for transferring at stations in the optional feed file *transfers.txt* (see [4]). Thus, we have information whether (1) it is possible to transfer at a station at all, (2) the departing vehicle is expected to wait for the arriving one with sufficient time for a passenger to transfer or the transfer buffer is given in order to change vehicles. Concerning the time-expanded graph we can model additional transfer rules, for example, for a given transfer buffer for a station, by setting the time of the transfer nodes belonging to that station accordingly. In the time-dependent model we handle transfer buffers in the cost functions. Hence, for a given transfer buffer TB_A in *transfers.txt* for station A where A has an edge to station B, we set the transfer buffer value at the edge (A, B) accordingly.

Contracting Departure Nodes In our experiments (see subsection 5.2.2), the size of the time-expanded graph needs about 6.5 times more memory as the time-dependent graph. We can reduce the size of the time-expanded graph by contracting

all departure nodes. As departure nodes have incoming edges from arrival and transfer nodes, we have to redirect those edges and adapt the cost of the edges accordingly. Contracting all departure nodes, we reduce the number of nodes and edges in the graph and hence the size of the time-expanded graph.

Transfer Patterns The experiments show that calculating the shortest paths on a network comprising the state New York using single and bi-criteria optimizations takes some seconds in the time-expanded and the time-dependent model. In order to reduce the query time we can use transfer patterns [1] which is an algorithm designed for transit networks. The idea is to precompute for each pair of stations all transfer patterns of all optimal paths and store them. A transfer pattern of a path from station A to B could be, for example, $A - B$ for a direct connection without transferring in-between and $A - C - B$ where we transfer at C. Then when we assume to have the timetable information for each station that allows us to determine very quickly the next direct connection from A to B, from A to C and from C to B, it is easy to answer the query $A@t \rightarrow B$ for an arbitrary time t.

Bibliography

[1] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast routing in very large public transportation networks using transfer patterns. *ESA 20120, Part 1, LNCS 6346*, pages 290–301, 2010.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Google Developers. Example gtfs feed. https://developers.google.com/transit/gtfs/examples/gtfs-feed. visited: April 7th,2012.

[4] Google Developers. General transit feed specification reference. https://developers.google.com/transit/gtfs/reference. visited: April 7th,2012.

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

[6] Geofabrik. Download openstreetmap extracts. http://download.geofabrik.de/osm/north-america/us/. visited: April 11th, 2012.

[7] R. P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9):670–676, 1983.

[8] MTA. Developer resources download. http://www.mta.info/developers/download.html. visited: April 11th, 2012.

[9] M. Müller-Hannemann and K. Schnee. Finding all attractive train connections by multi-criteria pareto search. *Railway Optimization 2004*, LNCS 4359:246–263, 2007.

[10] M. Müller-Hannemann, F. Schulz, D. Wagner, and C.D Zaroliagis. Timetable information: Models and algorithms. *4th Workshop on Algorithmic Methods for Railway Optimization (ATMOS 2004)*, pages 67–90, 2004.

[11] M. Müller-Hannemann and K. Weihe. Pareto shortest paths is often feasible in practice. *Algorithm Engineering WAE 2001*, LNCS 2141:185–197, 2001.

[12] M. Müller-Hannmann, M. Schnee, and K. Weihe. Getting train timetables into the main storage. *2nd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2002)*, Electronic Notes in Theoretical Computer Science, vol. 66. Elsevier, New York, 2002.

[13] openstreetmap.org. Elements. http://wiki.openstreetmap.org/wiki/Elements. visited: April 7th,2012.

[14] openstreetmap.org. Node. http://wiki.openstreetmap.org/wiki/Node. visited: April 7th,2012.

[15] openstreetmap.org. Openstreetmap. http://www.openstreetmap.org. visited: April 7th,2012.

[16] openstreetmap.org. Way. http://wiki.openstreetmap.org/wiki/Way. visited: April 7th,2012.

[17] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *Technical report, DELIS Project, Universität, Paderborn, Germany*, Submitted to Journal of Experimental Algorithmics, 2004.

[18] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Experimental comparison of shortest path approaches for timetable information. *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments*, pages 88–99, 2004.

[19] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards realistic modeling of timetable information through the time-dependent approach. *Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003)*, 92 of Electronic Notes in Theoretical Computer Science:85–103, Elsevier,2004.

[20] E. W. Weisstein. Great circle. http://mathworld.wolfram.com/GreatCircle.html. visited: April 7th,2012.