An Intermodal Travel Information System

Student research project
by Felix Gündling

Supervisors:
Dr. Mathias Schnee
Prof. Dr. Karsten Weihe

Fachgebiet Algorithmik
Technische Universität Darmstadt

April 2014

An Intermodal Travel Information System
Studienarbeit

Eingereicht von Felix Gündling
Tag der Einreichung: 30. April 2014

Gutachter: Prof. Dr. Karsten Weihe
Betreuer: Dr. Mathias Schnee

Hiermit versichere ich, die vorliegende Studienarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

30. April 2014 Felix Gündling

Contents

Chapter 1

Introduction

Most timetable information systems available today allow for finding train connections which are optimal regarding multiple criteria. All resulting journeys offered to the traveler start at a departure station and end at an arrival station. But this does not cover the full use case of most users. In practice, most journeys do not start at a station. This also applies to the destination: most users don't just want to visit a specific station but have a destination address they want to reach. Consequently, the user would like to enter two street addresses or even pick two coordinates from a map as an input for a travel information system. The system then should figure out an optimal connection between these two locations.

Thus, a possible query could look like this:

> "Search for a connection from Hochschulstraße, Darmstadt to Wilhelm-Busch-Straße, Bayreuth at 13:00 o'clock. I either walk or take a taxi at the beginning. At the destination I would like to walk."

Finding optimal answers to this kind of question is the main topic of this thesis.

Chapter 2

The Time Dependent Graph Model

For our approach to an intermodal travel information system we use a Time-Dependent (TD) graph model to perform a multi-criteria shortest path search on it in order to find Pareto-optimal connections.
Another structure that allows for searching train connections is the Time-Expanded (TE) graph model which basically has a node for each arrival and for each departure. Every departure is connected with its corresponding arrival by a directed edge. Public transport with a high frequency, such as busses and street cars lead to a very large amount of nodes that cannot easily be handled by normal desktop hardware. Therefore, we decide to build upon an existing, TD based prototype (named "TD") which is capable of handling high frequent public transportation.

2.1 Model

In the Time-Dependent model there basically exists exactly one node per station. Stations are connected by edges if and only if there exists a transport (e.g. a train) running between them. The edge-weight represents the costs of using this edge. Thus, it is not possible to assign a fixed weight to an edge. Instead, an edge in the Time-Dependent graph has a function attached that takes the current time as input argument and returns the edge cost. For the time criterion, this is the sum of the waiting time until the next train departure on this edge and the travel time to the station at the head of this edge.

2.2 Multi Criteria Shortest Path

The goal is to find connections that are optimal (minimize each criterion). An optimal connection is a connection that cannot be dominated by any other connection in terms of the following domination rules:
A connection dominates another connection if:

- it is at least as good or better regarding every criterion.
- there exists at least one criterion for which it is strictly better.

To find these connections, an extended version of Dijkstra's algorithm can be used. In order to support multiple criteria, it is not sufficient to assign a single value to each node. Instead, labels containing a vector of values are used. Since these labels can be incomparable (both are optimal - example: one has a short travel time but many transfers and another has a long travel time but no transfers), there can be many labels attached to one node.

At the beginning, the algorithm creates a start label at the source station node and puts this label into the queue. In every iteration, the first step is to extract one label from the queue. For each outgoing edge of the node this label is located at, a new label is created at the head node of this edge. If another label at the same node dominates the newly created label, this new label is removed. If that is not the case, all labels (at the same node) dominated by this new label get removed and the new label is added to the queue. The search terminates when the queue is empty.

2.3 Extending TD

In order to enable TD (mentioned at the beginning of this chapter) to be actually used in conjunction with a frontend, it is neccessary to write input and output methods. In this case, we adhere to the MOTIS XML format. This way, it is now possible to use TD with different frontends that where developed at the Algorithms group of the Technische Universität Darmstadt:

- **RaNDM**, which is a web frontend for the MOTIS timetable information system
- **MuMo** [Web11], which is a system (including a web frontend) which uses the MOTIS timetable information system and other services such as the Google street routing service in order to find intermodal connections

To be able to test the system, a tool was written to incorporate the footpaths of one timetable input data set into another.

Chapter 3

Intermodal Search

3.1 Definitions

The following terms will be used:

- The starting point of the journey: **source location**
- The final destination: **destination location**
- The first train/bus/streetcar station: **departure station**
- The last train/bus/tramway station: **arrival station**

3.2 Means of Transportation

For this thesis a journey has the following structure:

- The first part is the way from the source location to the departure station.
- In the second part only public transportation will be used to get from departure station to the arrival station.
- The last part is the way from the arrival station to the destination location.

The user specifies which means of transportation he wants the search algorithm to consider for the first part and for the last part.

For now, we consider the following possibilities: walking, cycling, using the own car, getting a lift by someone else, and using a taxi. Further possibilities would be to integrate bike sharing and car sharing solutions. These are currently being investigated at the Algorithms group of the Technische Universität Darmstadt.

Chapter 4

Introducing Prices

In a first approach we only considered the travel time and the number of transfers as search criteria. Since only the traveltime is important for the first and the last part of the journey, only the fastest alternative would be presented to the user (preferably using the car if available). One approach to prevent this is to introduce a domination rule where every means of transportation gets a value assigned: for example $walk = 1$, $bike = 2$, $car = 4$, $taxi = 8$. A connection can only be dominated by another connection with a value which is less or equal. This makes sense for the first phase of the journey. This way, the taxi connection cannot dominate the bike connection. In order to apply this approach to both, the first and the last part of the journey, it would be possible to add the corresponding values. This way, it is possible to prevent a connection $(car, ..., car)$ from dominating a connection $(walk, ..., walk)$, because $car + car = 8 > 2 = walk + walk$.
But in reality it is hard to justify why a connection $(car, ..., car) = 8$ where the total time in the car is 1h should be better than a connection $(bike, ..., taxi) = 10$ including a ten minute taxi drive and ten minutes cycling. Even when adjusting these values to fit certain cases, the approach does not yield reasonable results for many combinations.
This leads us to think about the reasons why we want to prevent the domination in the first case described above. For most users, this is the price: using a car or taxi is much more expensive than walking. Consequently, introducing the price as a new criterion solves this problem. Another reason could be environmental issues. But since there is no data available regarding the CO_2 emissions, we stick with the price criterion.
Introducing prices does not only concern the first and last part of the journey. Also the public means of transportation are not free of charge.

4.1 Distance Based Prices

Since using real prices for this search is not feasible, we calculate artificial prices based on the used means of transportation and the distance. Prices of the Deutsche Bahn are limited to a maximum of 140 euro. This limit also applies to our artificial prices when searching.

Private means of transportation
We consider walking and cycling to be costless. We assume a base price of 2.50 euro and a price

of 1.80 euro per kilometer for using a taxi. The price of driving an own car used in our search will be 30 euro cent per kilometer. Consequently, the price for getting a lift by someone else is two times 30 euro cent because in most cases the other person needs to drive back home.

Public means of transportation
For public means of transportation we consider different classes:

- High-speed trains cost 20 cent per kilometer.
- Long-distance trains cost 18 cent per kilometer.
- Local trains, busses, and streetcars cost 15 cent per kilometer.

4.2 Special Ticket for Fast Trains

Furthermore, additional charges for the high-speed and long-distance trains yield a more realistic model of the pricing applied by Deutsche Bahn.
The high-speed trains have an additional base price of seven euro. This base price gets added to the price of the connection the first time a high-speed train gets used. The long-distance trains have an additional base price of six euro which will be ommited if the base price of the high-speed trains was already payed. Moreover, the additional high-speed train charge will be only one euro if the charge for the long-distance train usage had already been payed.

4.3 Special Ticket for Local Transportation

To be able to represent the situation in Germany regarding the low-price segment, we introduce a ticket wich has a fixed price and allows the customer to use all regional trains in a fixed time-interval (i.e. one day or the whole weekend). This ticket does not apply to busses and streetcars because those providers are not involved.
Thus, we limit the total price of regional trains to a maximum of 42 euro. If the distance based ticket price exceeds this limit, the traveler would buy the special ticket (“Quer-durchs-Land-Ticket” / “Schönes-Wochenend-Ticket”).

4.4 Representation

The requirements mentioned above can be fulfilled by creating four price slots (integer values in euro cent) for each label:

- The first slot contains the price for high-speed trains (including the base price)
- The second slot contains the price for the long-distance trains (including the base price)
- The thrid slot contains the price for regional trains
- The last slot contains the price for busses and streetcars

4.5 Price Heuristic

To be able to apply the goal-directed search using lower bounds as described in [Dis07, p. 9] a lower bound graph for prices is required. This can be built using the cheapest distance based prices. A possible lower bound for the total price (in euro cent) of a label generated while searching can then be calculated the following way: $slot1 + slot2 + slot3 + max(4200, slot4 + heuristic)$ where *heuristic* is the lower bound retrieved from the corresponding graph. This value represents a lower bound, because it is not possible to travel cheaper than using only regional trains and using the special ticket for these trains if the price in *slot*4 reaches 42 euro. The value of 42 euro is a variable that can easily be adjusted in the program code.

Chapter 5

Searching For Connections Between Multiple Stations

This section first analyzes a very simple approach to solve this problem. After showing that this approach is not feasible for most queries, we'll introduce techniques which actually lead to an algorithm capable of solving the task in a decent time.

5.1 Simple Approach

A very simple approach used in [Web11] is to execute one query for each combination of possible departure and arrival station. This needs to be done for every means of transportation taken into account to get from the source location to the departure station and from the arrival station to the destination location. When considering a maximum of three means of transportation (e.g. walking, using the own car, and calling a taxi), with N stations in the region of the source location and M stations near the target location, this technique requires a maximum of $3N \times 3M$ requests in total.[1]
Since we are using a timetable containing local transportation like busses and streetcars, the number of stations that need to be considered can be very large, especially in urban areas where the density of bus stations is typically very high.
For example a query from Berlin to Frankfurt there are about 1500 stations in Berlin and about 1400 stations in Frankfurt that need to be considered when allowing a car ride of 20 minutes. This leads to 2,100,000 queries for a common query when considering only one means of transportation (car). This makes the simple approach practically infeasible.

5.2 One Label per Means of Transportation

The example above shows that it is neccessary to reduce the complexity of a query. One important factor when searching are the different means of transportation that can be used to get to the differen departure stations. Let us consider the following query: For the first part (getting

[1]A schedule without busses and streetcars was used. So this approach seemed promising at the time

from the source location to the departure station) the user considers either to drive 20 minutes using his own car or take a taxi (tradeoff: paying more money but saving the time to search for a parking lot). Thus, we need to have two searches for each station in a "20 minutes radius" (containing all stations reachable within 20 minutes) around the source location.
Instead of applying two complete searches, the number of searches can be halfed by placing two labels for each departure in the start interval at the start station and putting them both into the priority queue before starting the search. Therefore, at this stage we have to do $N \times 3M$ searches which is an improvement of a constant factor of one to three (depending on the search query).
This approach can now be further improved by putting all start labels generated at all start stations into the priority before starting the search. Consequently, the effort can be further reduced to $3M$ searches. These searches are much more sophisticated because they start with much more start labels in the priority queue which results in longer query times. This method reduces the number of required queries from 2,100,000 to 1,400 which is already a big advantage.

5.3 Labels Containing More Than One Start

Because labels are costly, it makes sense to reduce the number of labels created at the beginning. Let us analyze the different start labels created at a station: For each departure there are several start labels (one for each means of transportation). These labels only differ in the criteria vector. All other features (node they are attached to, train departure time, train prices, and interchange count) are the same. This discovery can be used to join these labels into one and only store the different starts. Therefore, the same datastructures can be used to hold the criteria (travel time, transfers, price) of the middle part of the journey (public transportation) and only one additional array holding the different start times and prices is required. This way, one label can represent many connections: each connection has a different first part.
In order to apply the domination rules to labels containing different starts, it is required to iterate every start of the first label and compare the resulting connection (as a whole) with every connection represented by the second label. If one connection of the first label dominates a connection from the second label, the second connection needs to be marked as invalid. Thus, an additional boolean flag is required to indicate whether a connection is active or not. This can also be used to deactivate a certain slot at the beginning, in case it's not in use (i.e. if a station is only reachable by car, the other slots will be deactivated). Consequently, a label can only dominate another label if all slots of the second label are set to inactive.

5.4 Introducing Virtual Edges at the Target

After the optimizations of the last two sections solving the problem still requires $3M$ searches, one for each target station.

5.4.1 Simple Approach

A first approach would be to create a meta station representing all target stations because the concept of a metastation is already available in TD. This way, the search accepts every every

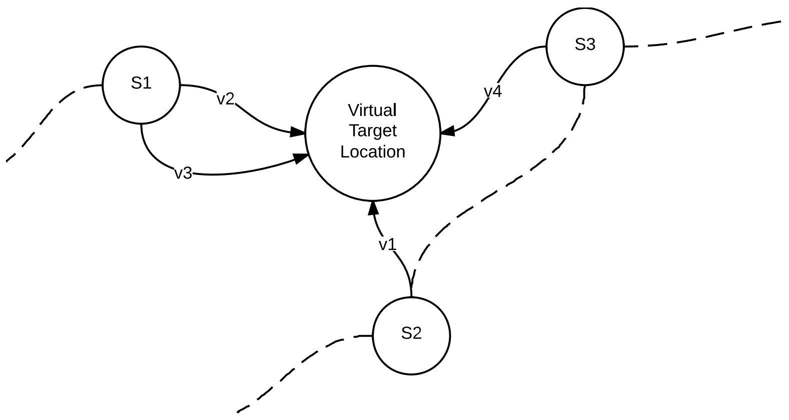

Figure 5.1: Introduction of new virtual target node and virtual edges connecting all arrival stations with the target location: The nodes S1, S2, and S3 represent the stations which are considered potential target stations. The edges v1 to v4 connect these nodes with a dummy target node that has been added to represent the target location.

label placed at any target station as terminal-label and uses it for domination by terminal [Dis07, p. 9]. But since this would imply that a label at any target station can be compared to any other label that has reached another target station, we would neglect the last part of the journey. This can lead to very confusing results because this way the seach will often avoid stations close to the target and tend to find connections with a target station which can be reached from a source station in a short time. So this approach does not yield desirable results.

5.4.2 Virtual Edges and Additional Target Node

Another approach is to create an additional target node representing the real destination location. To incorporate this new node into the railway graph, new edges are required. These virtual edges have the same behavior as footedges: They have fixed edge costs and can always be used (as opposed to train edges which can only be used at certain times). For each means of transportation considered to reach the destination location from an arrival station, a new virtual edge connecting the (potential) arrival station with the destination location gets added to the graph. The edge weights are equal to the required time and the costs calculated as input for the search algorithm. Using one of these virtual edges does not increase the transfer count.
Figure 5.1 shows a simple example. There, we can see a virtual target node representing the destination. It is connected to all potential target stations. For example, it is now possible to reach the destination location via arrival station S1 using either v2 or v3. So there are two different means of transportation (i.e. taxi and walking) available to get from S1 to the destination location. For station S2 and S3 there is one means of transportation available. Additionally, it's possible to reach S3 from S2 and not take the direct edge v1 connection S2 with the virtual target node. This can make sense if for example v1 is a foot edge taking a long time (but for free) and the edge (S2, S3) and v4 are more expensive but faster.
This way, it is now possible to have a complete search from all departure stations to the target

node which does not require any special rules besides the default algorithm described in Chapter 2. Hence, the complexity of solving the problem can now be reduced to exactly one search.

5.4.3 Lower Bound Graph Adjustment

To be able to apply speed up techniques described in [Dis07, p. 9], it is required to have valid lower bounds for each node of the graph. Since the search graph was changed significantly in the last step, it is now required to also adjust the graphs used to caclulate the lower bounds. This can be done easily by adding the same virtual target node and edges connecting the potential arrival stations with this target node as described in the last section. Consequently, when applying the search on the adjusted lower bound graph, we obtain valid lower bounds for all nodes.

Chapter 6

Routing Service

Figure 6.1 shows the basic structure of the routing process. In the following, we will walk through the steps that are required in order to serve a user request. Appendix A.1 and A.2 describe the query and response format for the service.

1. First, the user selects two coordinates in a frontend. In addition to that, he must specify which means of transportation he would like to use at beginning and at the end of his journey. Based on this input, the frontend then generates a JSON request which follows the structure described in Appendix A.1 and sends it to the service URL via HTTP.

2. The HTTP frontend accepts the request and deserializes the query, generating a datastructure that then gets passed to the routing engine.

3. The following steps need to be executed for both, the source and the target: The routing engine calculates the radius from the maximal average speed and the maximal duration for each means of transportation. Then, it queries every station which is within these radii from the station geo index. This way, there is one set of stations for each means of transportation available. These sets can be empty (no station can be reached using this means of transportation).

4. The generated sets of stations are the basis for creating queries for the OSM routing services. In our case we are using the Open Source Routing Machine (OSRM)[1] which provides a HTTP interface accepting a source and a target location. As indicated in Figure 6.1 any other service which is compliant to the OSRM interface can be used. These requests then get executed in parallel. The responses contain (among others) the duration (it takes to get from the source location to the departure station / from the arrival station to the destination location) and the distance in meters.

5. Based on the responses from the OSM routing services (containing the travel duration for the private transportation part of the journey) a filtering gets applied which removes the instances where the maximal duration for a specific means of transportation is exceeded.

[1]http://project-osrm.org/

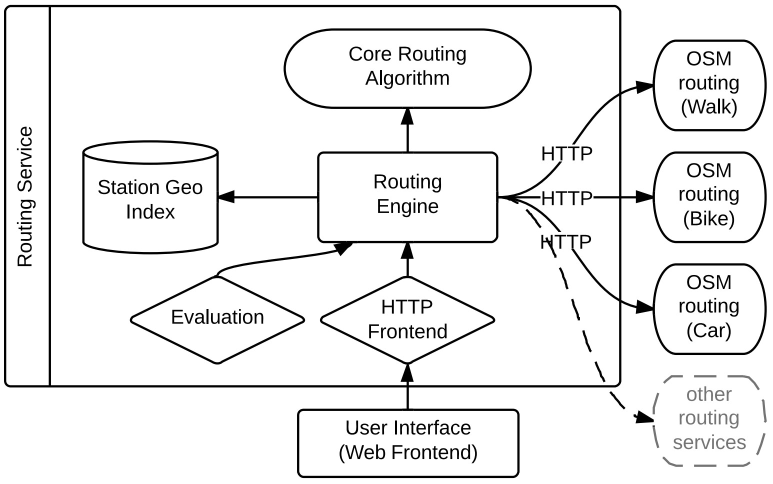

Figure 6.1: The basic service structure showing the different components of the routing: The central routing engine handles requests generated by the HTTP front end or the evaluation. In order to serve these requests, it's using a station geo index to determine stations located within a specified radius. Furthermore, a street routing service calculates durations and distances for the non-public transport parts of the journey. Finally, the core routing algorithm described in Chapter 5 calculates the complete journey.

6. After this, the routing engine generates a query for the routing algorithm developed in Chapter 5 and executes it. The result then gets serialized using the XML format described in Appendix A.2 and returned to the routing engine.

7. The serialized XML response then gets sent back to the client which parses and visualizes the response.

Chapter 7

Performance Evaluation

7.1 Setup

To evaluate the performance of the algorithm developed in this thesis, we generate random intermodal queries based on real timetable queries (station to station) provided by Deutsche Bahn. To produce two coordinates (source and destination location) that are not equal to exactly the coordinates of a station (like in the original queries), the following process was applied:

- Read the EVA number of the source and destination station and try to find the corresponding station in the current schedule data. If no match could be found, the query is discarded.
- Generate a random coordinate within a 30km radius around the source/target station. This will be the source/destination location.

Since the source queries are all within one week (2007-08-13 - 2007-08-19), their query dates were mapped to our current schedule (2014-01-23 - 2014-01-27).

Table 7.1[1] shows the means of transportation used in order to randomly generate the corresponding query settings.

[1]"Entfernungspauschale" 30 cent/km from §9 Abs.1 Nr.4 und Abs.2 EStG

Name	Max. Duration	Max. avg. speed	Penalty	Base Price	Price per km
Own car	15	80	8	0	30
Other car	15	80	0	0	60
Taxi	15	80	0	250	180
Bike	15	15	5	0	0
Walk	15	5	0	0	0

Table 7.1: Considered means of transportation: Times in minutes, prices in euro cent. "Other car" = getting a lift up by somebody else. See Section 3.2 for more information.

In order to evaluate the performance of the algorithm with queries of different complexity, we generate three query sets. These query sets differ in the means of transportation that are allowed to get from the source location to the departure station. The query generator selects randomly from the allowed combinations:

- The first set only allows “Own car” or “Bike” or “Taxi”.
- The second set only allows one of the following combinations:
  - “Walk” and “Own car”
  - “Walk” and “Other car”
  - “Walk” and “Taxi”
  - “Bike” and “Own car”
  - “Bike” and “Other car”
  - “Bike” and “Taxi”
  - “Own car” and “Taxi”
  - “Other car” and “Taxi”
- The third set only allows one of the following combinations:
  - “Walk”, “Own car”, and “Other car”
  - “Bike”, “Own car”, and “Other car”
  - “Walk”, “Own car”, and “Taxi”
  - “Bike”, “Own car”, and “Taxi”

Each set contains 500 queries. So in total this evaluation has 1,500 queries.
The set of allowed means of transportation that can be used to get from the arrival station to the destination location is randomly picked from the following combinations:

- “Walk”
- “Walk”, “Other car”
- “Walk”, “Taxi”
- “Other car”
- “Taxi”

7.2 Results

The following results were obtained by executing the queries described in the last section on a computer with a Intel(R) Xeon(R) CPU E3-1245 V2 CPU (4 cores), running at 3.40GHz. The machine has 32GB of main memory. This allows us to run two instances in parallel (peak memory usage: about 11GB each).

We evaluate three different versions:

- The "full" algorithm as described in Chapter 5.
- Labels with only one single slot (as described in Chapter 5 until Section 5.2). In the following, we will call this approach "multi-label".
- One search per departure station (still keeping the virtual destination node).

In the following, we will use the term "slot" as a short term for one means of transportation that can be used to get from the source location to the departure station.

As shown in Figure 7.2 the average search time was 4,662ms for one slot, 6,355ms for two slot and 5,822ms for three slot queries using the full search. The multi-label approach performed better for the one slot and two slot queries (4,324ms, and 5,892ms) but was slower on the three slot query set (6,311ms). We assume that the full search approach would outperform the multi-label approach for queries with more than three slots.

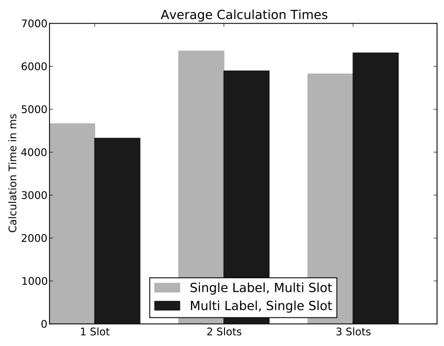

Figure 7.1: The multi label (each with one single slot) search performs better on the one slot (338ms faster) and two slot (463ms faster) searches. The multi-slot search is 489ms faster for queries with three slots.

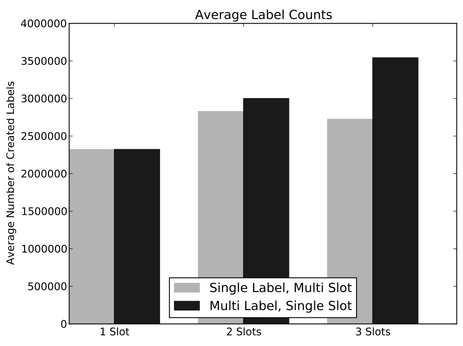

Figure 7.2: For search queries with two start slots, the multi-label search approach requires 20,651 more labels. For queries with three start slots 106,448 more labels are required.

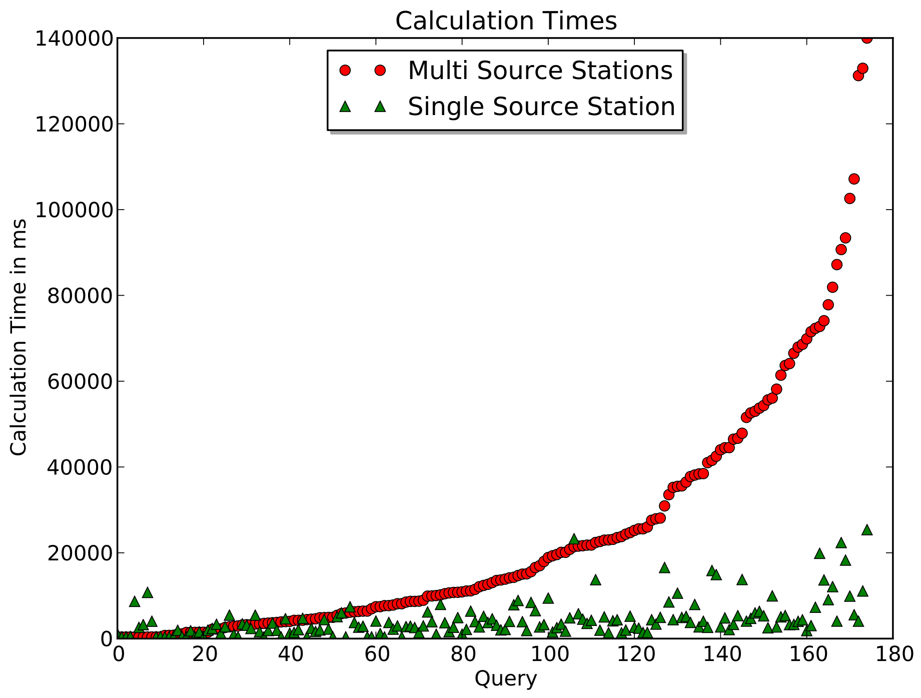

Figure 7.3: Comparison of the single-source and multi-source search approach for queries with less than 35 source stations. Searches are ordered by the calculation time of the single-source search.

Figure 7.3 shows a comparison between the multi-label search algorithm and the approach where we did one search per source station. Only queries with less than 35 potential departure stations were executed. As we can see, there are some searches where both algorithms require less than 10 seconds and don't differ much in the required calculation time. But there are many more queries where the simple approach (one search per departure station) requires much more time than the multi-source search. On average, the multi-source approach required 4,442ms and the single-source approach took 24,298ms. So the speed-up factor is about 5.47. Since only "simple" queries with less than 35 departure stations were evaluated, we can assume this factor to be even greater for more complex queries.

Chapter 8

Conclusion

In this thesis, we presented an algorithmic approach to efficiently search for intermodal connections considering multiple criteria like travel time, price and transfers. The number of searches required to solve the problem in an optimal way was reduced from $N \cdot 3 \times M \cdot 3$ (N, M is the number of stations at the start/target) to a single search. This involves multi-source and multi-target search extensions to an existing Time Dependent search algorithm which is capable of handling high-frequency local traffic like busses or streetcars.
Furthermore, a complete routing service (connected to Open Streetmap Routing services for non-public means of transportation routing) including a web front end prototype was developed showing that the search algorithm is not only a theoretical approach but can actually be put to use.

Chapter 9

Future Work

9.1 Direct Connections

9.1.1 Problem

In some cases, the search produces results that are not very useful. For example, if the car radius around the source location already contains the destination location, the fastest connection of the result list is often to drive to a station that's very close to the target, then drive a few meters with public means of transportation and then walk the rest to reach the target. This is an obvious indicator that the user should consider driving directly from the source location to the target location. This can also happen with other means of transportation like cycling or even walking if the source location and destination location are very close.

9.1.2 Solution

To improve the search, it would be useful to consider the direct connection in those cases. A very simple (and obvious) indicator could be, a radius (generated by a corresponding means of transportation) around the source location containing the destination location. In this case, the routing engine should query the direct connection (from source location to destination location) for this specific means of transpotation.
Another approach would be to also consider cases where the first condition is not true but for example the car radius around the source location and the car radius around the destination overlap. This is also a clear indicator that the search should consider the direct car connection. A more difficult case could be where the a car radius at the source location overlaps with the bike radius at the destination location.
Another approach, requiring user interaction, would be to add an extra checkbox option to the frontend enabling the user to query the search engine to consider direct connections with non-public means of transportation.

9.1.3 Algorithmic Implementation

To be able to compare the direct connection to all other connections with respect to duration, price, and transfers, this direct connection also needs to be added to the graph. This could be accomplished by adding a virtual source node that's connected to all potential departure stations. This is the same approach as described in Section 5.4 for the destination node. This way, it would be possible to add a virtual edge connecting the virtual source node and the virtual destination node. The edge weights of this edge represent the costs of the direct connection(s) calculated by the routing services for non-public means of transportation.
The direct edge from the start node to the target node will quickly generate a label at the target station that probably has a very short travel time. Consequently, this extension would probably also speed up the search (see [Dis07, p. 9] - 2.3.3 Dominance by Early Results).

9.2 Consider More Means of Transportation

In our approach we didn't consider using car sharing and bike sharing alternatives. But since these solutions are gaining a lot of popularity these days, it definitely makes sense to include them in an intermodal routing engine.

9.3 Realistic Penalty Times

For our current implementation, we only used fixed times to account for the times required to search for parking lots. This can be improved by distinguishing between urban areas where it is hard to find a free parking lot and countryside where it is usually easy. Using a database containing information about public parking capacities, it would be possible to produce results that are significantly more realistic.

Appendix A

Appendix

A.1 Query Format

The query object shown in Figure A.1 sets a time interval by defining a start time and the length of the interval. `date_time` has the format `yyyy-mm-ddThh:mm`. So ten o‘clock at the 30th of April 2014 would be formatted this way: `2014-04-30T10:00`. Together with the `interval` value (in minutes), this defines the query interval. In addition to that, the `source` and `destination` attributes define the location of source and destination and the means of transportation that should be taken into consideration.

```
{
  'date_time': ${query_time:string},
  'interval': ${query_interval:int},
  'source': ${source_definition:object},
  'destination': ${destination_definition:object}
}
```

Figure A.1: Overall query format defining the start interval time and definitions for source and destination described in detail in Figure A.2

For the source and the destination description, the format shown in Figure A.2 is used. The `location` attribute defines the latitude and longitude of the source/destination. The name is not required for routing but can be used for humans to identify the location more easily. Additionally, the `arrivals` attribute describes a list of means of transportation that should be taken into consideration.

```
{
  'location': {
    'lat': ${latitude:double},
    'lng': ${longitude:double},
    'name': ${station_name:string}
  },
  'arrivals': ${arrivals:array}
}
```

Figure A.2: Definition setting the location and possible arrivals of source/destination. Detailed format of arrivals is described in Figure A.3

For the arrival description (shown in Figure A.3) the `name` attribute is used to determine the routing service that should be queried. The attributes `max_duration` and `max_avg_speed` define the search radius for stations to route to. Furthermore, the `max_duration` is used to apply a final filtering on the street routing results. The `onetime_penalty` value (in minutes) will always be added to the travel duration of this means of transportation. This can be used to take the time that is required to search for a parking lot into account. The `time_multiplicator` can be used to adjust the duration values returned by the routing service. Thus, walking or cycling times can be adjusted to suit elderly persons. The `price` has two components: A kilometer based price (`per_km`) and a base price (`base`). This can be used to model the taxi pricing.

```
{
  'name': ${name:string},
  'max_duration': ${maximal_duration:int},
  'max_avg_speed': ${maximal_average_speed:int},
  'onetime_penalty': ${one_time_penalty:int},
  'time_multiplicator': ${time_multiplicator:double},
  'price': {
    'base': ${base_price:int},
    'per_km': ${price_per_km:int}
  }
}
```

Figure A.3: Description of a possible arrival at a station: This defines the name (used to discover the routing service to use)

A.2 Response format

The following snippet of XML encoded text shows an example response produced by the routing webservice:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataExchange>
   <ConnectionList>
      <Connection>
         <StopList>
            <Stop evaNo="-1" name="DUMMY" lat="49.8782" lng="8.65452">
               <Departure dateTime="2014-01-23T06:44" />
            </Stop>
            <Stop evaNo="0124747" name="Willy-Brandt-Platz, Darmstadt" lat="49.8761" lng="8.65053">
               <Arrival dateTime="2014-01-23T07:40" platform="-" />
               <Departure dateTime="2014-01-23T07:40" platform="unbekannt" />
            </Stop>
            [...]
            <Stop evaNo="0115993" name="Pfälzer Schloß, Groß-Umstadt" lat="49.8676" lng="8.92673">
               <Arrival dateTime="2014-01-23T08:25" platform="unbekannt" />
               <Departure dateTime="2014-01-23T08:36" platform="unbekannt" />
               <InterchangeInfo/>
            </Stop>
            [...]
            <Stop evaNo="0123019" name="Wald-Amorbach Volksbank, Breuberg" lat="49.8482" lng="9.02653">
               <Arrival dateTime="2014-01-23T09:17" platform="-" />
               <Departure dateTime="2014-01-23T09:17" platform="unbekannt" />
            </Stop>
            <Stop evaNo="-1" name="DUMMY" lat="49.846" lng="9.0211">
               <Arrival dateTime="2014-01-23T09:21" platform="-" />
            </Stop>
         </StopList>
         <JourneyInfo>
            <Walk price="0" from="0" duration="56" to="1" slot="walk" />
            <Transport name="Bus 671" categoryName="Bus" from="1" to="19" />
            <Transport name="Bus K68" categoryName="Bus" from="19" to="29" />
            <Transport name="Bus K68" categoryName="Bus" from="29" to="30" />
            <Walk duration="4" from="30" to="31" />
            <Walk duration="4" from="31" to="32" slot="bike" price="0" />
            <Attribute code="OB" text="..." from="1" to="19" />
            <Attribute code="FB" text="..." from="19" to="29" />
            <Attribute code="FB" text="..." from="29" to="31" />
         </JourneyInfo>
      </Connection>
   </ConnectionList>
   <Query>[...]</Query>
</DataExchange>
```

The format is based on the MOTIS XML format. It contains a list of connections in the `DataExchange` node. Each connection has a `StopList` defining a the sequence of stops that will be visited. The first stop and the last stop are always the source and destination location. Each stop has its unique EVA number identifier (`evaNo`) and its coordinates attached as attribute. Stops can contain an `Arrival` and an `Departure` tag containing information about the arrival time and departure time. Additionally these nodes contain the platform if available. Additionally, a stop node can be marked as interchange if it contains an `Interchange` node.

Furthermore, each connection contains a `JourneyInfo`. This node contains information about the transports used and attributes that apply for specific sections of the journey. The first and the last child node concerning the used transports are always `Walk` tags. The tag name does not represent the means of transportation but was choosen for compatibility reasons. The actual name of the means of transportation is mentioned in the `slot` attribute.

List of Figures

Bibliography

[Sch09] Mathias Schnee. *Fully Realistic Multi-Criteria Timetable Information Systems*. Dissertation, Technische Universität Darmstadt, 2009.

[Web11] Christian Manuel Weber. *Multimodal Real-Time Timetable Information - Design, Implementation, and Evaluation of a Prototype*. Master Thesis, Technische Universität Darmstadt, 2011.

[Dis07] Yann Disser. *Multi-Criteria Search for Optimal Train Connections using the Time-Dependent Graph Model*. Diploma Thesis, Technische Universität Darmstadt, 2011.