Master Thesis

Multi-Modal Route Planning

Mirko Brodesser

April 2013

University of Freiburg
Faculty of Engineering
Department for Computer-Science

Bearbeitungszeitraum
September 2012 – April 2013

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

Betreuer
Prof. Dr. Hannah Bast
Dr. Sabine Storandt

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 11. April 2013 Mirko Brodesser

Zusammenfassung

Wir untersuchen die Berechnung von optimalen Routen, die aus der Benutzung von öffentlichen Verkehrsmitteln, Laufen, und Autofahren bestehen. Eine eigene Herausforderung ist es, prägnante und vielfältige Mengen an Routen zu bestimmen. Wir betrachten Großstädte wie New York, bei denen eine schnelle Berechnung der Routen eine weitere Schwierigkeit darstellt. Wir veranschaulichen, dass mehrere Optimierungskriterien notwendig sind um vielfältige Mengen an Routen zu erhalten. Dies erweitert die Menge an optimalen Routen stark, wobei viele der Routen ähnlich sind. Um prägnante Mengen zu bestimmen, studieren wir verschiedene Filterungsmethoden. Wir stellen ein Filterungsverfahren, *Types aNd Thresholds (TNT)*, vor, was zu den gewünschten Mengen führt. Zur Reduzierung der hohen Berechnungszeit stellen wir optimalitätsbewahrende sowie heuristische Ideen vor. In einer experimentellen Untersuchung werten wir die gefundenen Wege bezüglich Qualität und Berechnungszeit aus. Die Experimente belegen die geforderte Qualität von TNT und zeigen, dass die Laufzeit, unter Inkaufnahme eines Bruchteils an suboptimalen Lösungen, in der Größenordnung von wenigen Sekunden liegt. Wir beenden dieses Papier mit einer Übersicht von Verbesserungsmöglichkeiten.

Abstract

We examine the computation of optimal paths, which consist of using public transportation, walking and taking a car. A challenge in itself is to determine concise and diverse sets of paths. We consider metropolises like New York, where a fast computation of the paths is difficult. We exemplify that multiple optimality criteria are necessary to obtain diverse sets of paths. This extends the set of optimal paths severely, whereas many paths are similar. To determine concise sets, we study different methods of filtering. We propose a filtering procedure, *Types aNd Thresholds (TNT)*, which leads to the desired sets. To reduce the high computation times, we introduce several optimality-preserving and heuristic ideas. We experimentally evaluate the found paths with respect to quality and computation time. The experiments show the claimed quality of TNT and that the running time, while accepting a fraction of suboptimal solutions, is in the magnitude of few seconds. We complete the paper by giving an outline of possible improvements.

Contents

1 Introduction

Travelling shorter or longer distances every day has become a habit for many people world-wide. Whether one has to get from home to work, from work to an external meeting, or simply to visit a foreign city, one has to travel. If we want to travel by car, we can find the shortest path using a navigation system. If we want to travel with public transportation, there are websites to look up available connections. However, if we want to travel somewhere, but do not want to limit ourselves to one type of transportation, currently existing retrieval systems are too restricted to help us. For example, there could be a path where taking a taxi between two stations yields a real advantage. Present systems miss this path. We focus on computing feasible sets of paths, which include such an advantageous path. We consider a set of paths feasible if it is small, concise and diverse. Such sets of paths are good to compare the truly distinct paths easily and allow to quickly decide for one of them.

As modes of transportation we consider walking, taking a car, public transportation, and their combinations. The corresponding networks we regard are almost unrestricted. Most importantly, we allow for car usage at arbitrary parts of journeys. To obtain diverse sets of optimal paths, we use multiple criteria: travel duration, car duration, walking duration, and the number of transfers. We exemplify that multiple criteria are necessary for diverse paths. This significantly extends the set of optimal paths, whereas many paths are similar. To get concise sets we propose several approaches to filter the numerous results obtained. In comparison to uni-modal optimal path calculation, computation times increase significantly. Therefore, we introduce optimality-preserving as well as heuristic methods to reduce them.

Taking into account the properties (availability, velocity, price) of the different means of transportation, we present a filtering procedure, *Types aNd Thresholds (TNT)* which leads to sets of paths with the above mentioned properties.

The text is structured as follows. In section 2, we give a brief outline over the existing work and findings in this area. In section 3, we recapitulate representations and outstanding algorithms for the different types of networks. We introduce a model combining the different types of networks to one multi-modal network and a method to calculate optimal paths on it. In section 4, we examine several combinations of optimality criteria and different filtering methods. The section is finalised with the introduction of TNT. Section 5 is about speed-up approaches. We proceed with the experimental evaluation of the proposed algorithms and their respective speed-up techniques in section 6. In section 7, we sketch possible improvements with respect to quality, efficiency and more realistic modelling and conclude with section 8.

2 Related Work

In this section we give a brief overview over existing algorithms in the area of shortest path computation. The basis of many state-of-the-art algorithms forms *Dijkstra's Algorithm* [1], presented in 1959. For every graph with non-negative edge values, it computes a shortest path from a given source node to all other nodes. For details, see section 3.2.1.

For road networks exist many algorithms to reduce the time of calculating a shortest path from one location to another. Among these are A^*, *Landmarks*, *Arc-Flags*, *Transit Node Routing* and *Contraction Hierarchies*. A^* [2] is a generalisation of Dijkstra's Algorithm. It allows the usage of optimality preserving heuristics to compute the distance to the target node with less iterations. One of these heuristics is to use *Landmarks* [3]. The idea is to choose a very small, diverse set of nodes, the landmarks. It is accompanied by the precomputation of the distances of the landmarks to all other nodes. For a query, the precomputed distances are used to estimate the distance of a node to the target. This leads to less iterations until the distance to the target node is known. Assuming many queries are performed, precomputation is a good investment. Another algorithm, *Arc-Flags* [4], precomputes a partition of the graph to regions. Each node is assigned to exactly one region. Each arc has a flag for each region r. The flag indicates if the arc has to be considered, when calculating the shortest path to a node in region r. This allows to skip many arcs and hence nodes, thus reducing the query time. A further approach that yields very low query times is *Transit Node Routing* [5]. The algorithm is based on the observation that the shortest path of long-distance queries always passes through a small set of nodes which are close to the source node. These nodes are called *transit nodes*. The same applies for the target node. Exploiting that the set of all transit nodes can be chosen small, makes precomputing their pairwise distances feasible. Using the precomputed results, long-distance queries can be answered efficiently. A major improvement in computing shortest paths are *Contraction Hierarchies* developed by Geisberger et al. [6]. We recapitulate the algorithm in section 3.2.2.

With respect to modelling, we consider transit networks as more complicated than road networks. For transit networks several models exist. We briefly introduce two of the most common ones, *time-expanded* and *time-dependent*, in section 3.1.2. Details can be found in [7]. For the time-expanded model, the same algorithms as for road networks can be applied. One approach for transit networks, introduced by Bast et al. [8], is to precompute *Transfer Patterns*. This method leads to query times of a few milliseconds, also for huge networks. We study this idea in section 3.2.3 in more

detail. Recently, a round-based approach that does not rely on precomputation was introduced in [9].

Predominantly for transit networks, multi-criteria optimisation receives growing attention. Among the considered criteria are total duration, car duration, walking duration, the number of transfers, and costs. Mapping multiple criteria linearly to one criterion is considered problematic [10], because some desired solutions are likely to be missed. Recent approaches focus on *Pareto Sets* [11] which lead to non-dominating optimal paths. For a detailed discussion, see section 4.

Some approaches for transit networks (like [8]) consider walking to nearby stations. Others consider different modes of transportation, restricted to a certain hierarchy [12]. Less restricted multi-modal networks have recently been considered by Braun [13] and Delling et al. [14]. The latter two papers show that using Pareto Sets with multiple criteria enlarges the set of optimal paths to an impractical extent, especially when car duration is considered. The paths become too numerous to be practical when showing them to a user. Moreover, many paths are very similar and query times increase to the order of minutes. Braun presented a model restricting approach which significantly reduces the size of the Pareto Sets. However, Braun considers his approach too restrictive with respect to preserving quality. An evaluation of the latter is missing. Delling et al. introduced an approach that uses techniques from fuzzy logic. It identifies the top-k most significant journeys in a post-processing step. However, their definition of significance does not necessarily result in a diverse set of solutions.

Modelling reliability and robustness (i.e., how good are the alternatives, in case a transfer is missed) are other important aspects. Recently, the work of Strasser [15] focused on this topic. However, it is limited to transit networks. In this paper, we do not focus on this aspect. In the following section, we recapitulate modelling aspects of road and transit networks and introduce a model to combine them to a multi-modal scenario. Moreover, we present a method to combine existing algorithms to compute optimal paths on the combined model.

3 Routing Models and Algorithms

3.1 Models

We distinguish two types of networks, road networks and transit networks. The former are in our case car and walking networks, the latter represent buses, subways, trains, etc.

3.1.1 Road Networks

A road network is modelled as a graph. Each node corresponds to a position on earth. Each arc (u, v) contains the duration it takes to directly travel from u to v. Figure 3.1 shows a simple example. Note that the graph is directed.

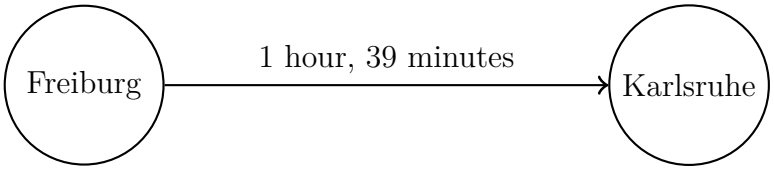


Figure 3.1: Example of a road network graph.

3.1.2 Transit Networks

Transit vehicles operate on fixed schedules. For each departure of a vehicle exists a next station where the vehicle stops at a certain time. We call this an *elementary connection*. For transit networks, two models are common: One is called **time-dependent**, the other **time-expanded**.

In the **time-dependent** model, each station of the real world corresponds to one node in the model. For each elementary connection, an arc exists between the respective nodes. Attached to the arc is a function depending on the time, mapping to the duration to get from one station to the other. The function is also used to model the waiting time until the connection is available. Figure 3.2 shows an example of a time-dependent graph. This model yields a compact representation of a transit network. However, including transfer buffers is not trivial. There are several possibilities to solve this issue; for details consider Pyrga et al. [7].

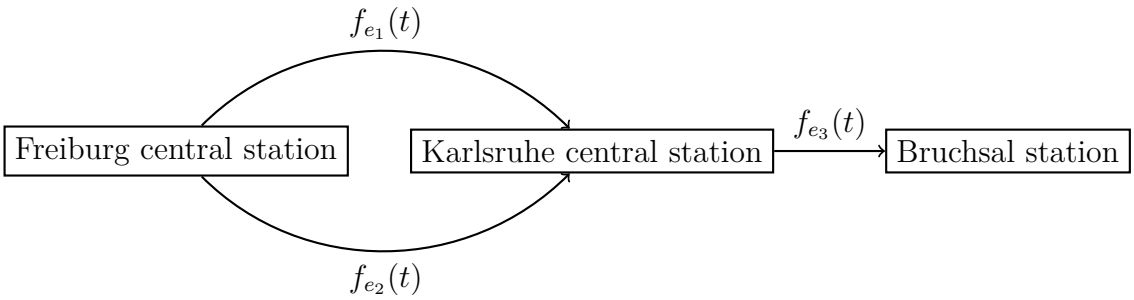


Figure 3.2: Example of a time-dependent graph. The elementary connections are denoted by e_i, the corresponding functions by f_{e_i}.

In the following we recapitulate the **time-expanded** model. For every departure and arrival of an elementary connection, there exists one departure and one arrival node. To model transfer buffers for each arrival and departure, a transfer node is added. For a detailed explanation, we again refer to Pyrga et al. [7]. Figure 3.3 shows an example of a time-expanded graph. Since transfer buffers are explicitly modelled in the graph, dealing with them is simple. Most algorithms applicable to road networks can also be applied to time-expanded graphs. The reason is simple, time-expanded graphs are also graphs with non-negative edge costs.

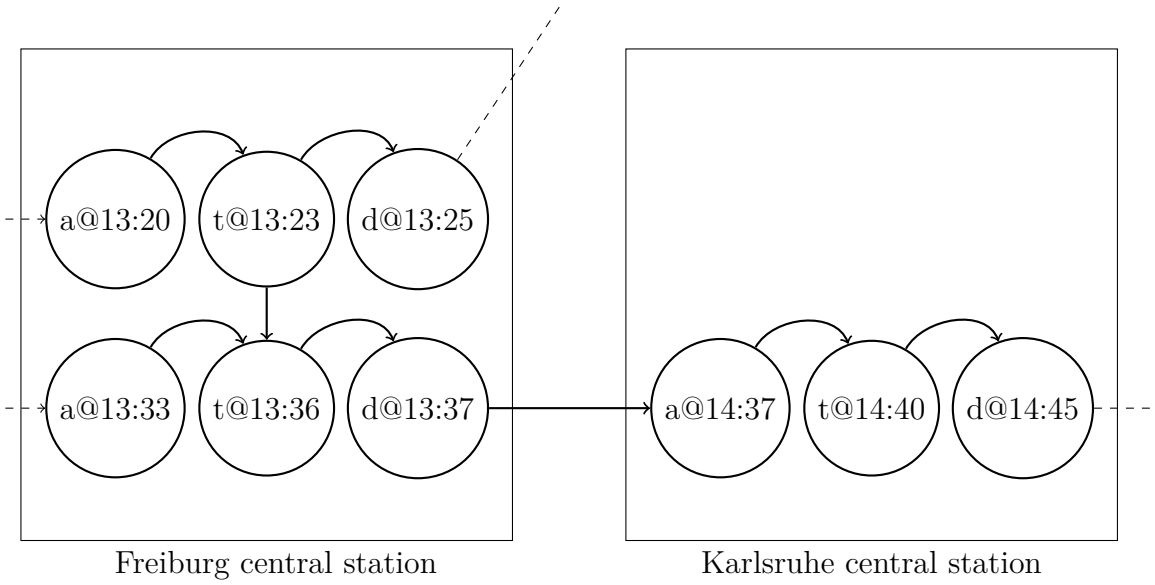


Figure 3.3: Example of a time-expanded graph. Each circle corresponds to one node. They are labelled with *type*@*time*, where type is either *a* (arrival), *t* (transfer) or *d* (departure). For simplicity, the durations of the arcs, the differences of the time of the two adjacent nodes, were omitted.

3.2 Algorithms

We briefly introduced some existing shortest path algorithms in section 2. In this section, we discuss one state-of-the-art algorithm for each type of network. For road networks, we introduce *Contraction Hierachies*, for transit networks *Transfer Patterns*. Before, we sketch the idea of Dijkstra and two of its relevant variants.

3.2.1 Dijkstra & Variants

Dijkstra's Algorithm (often abbreviated as *Dijkstra*), is a shortest path algorithm for graphs with non-negative edge values. For a given source node, it calculates the shortest paths to all other nodes. Initially, tentative distances to all neighbours of the source are kept in a priority queue (PQ). In each following iteration, the node with the currently smallest tentative distance is taken out of the PQ. The nodes tentative distance then equals its optimal distance to the source node. A node taken out of the PQ is called *settled*. Whenever a node is settled, the tentative distances of the neighbours are updated according to the edge values. This is called *relaxing* the edges. For the calculation of a shortest path from A to B, the procedure can be terminated as soon as B is settled. Given a graph $G = (V, E)$ and using a Fibonacci Heap, the amortized running time lies in $O(|E| + |V| \cdot log(|V|))$. The original algorithm presented by Dijkstra does not use a PQ and has a running time of $O(|V|^2)$. However, when referring to Dijkstra one usually means the optimised version which uses a PQ. We also stick to this habit.

Multi-Criteria Variant. There exists an extended version of Dijkstra, which takes into account multiple criteria in terms of Pareto Sets (see section 4.1 for details). Roughly, the idea is that instead of tentative distances, multiple tentative objects are kept per node. An object corresponding to a specific node is called *label*. Here, the PQ contains labels instead of integer values. For Pareto Sets, the PQ order has to be a refinement of the order for the criteria. In the following, the important point is that we can and do choose a PQ order where labels are popped from the PQ by increasing duration.

Bidirectional Variant. With respect to Dijkstra, Bidirectional-Dijkstra is used to reduce the number of visited nodes. It uses a graph with inverted arc directions, we call it *backwards graph*. Parallel to a Dijkstra on the graph beginning at the source, a Dijkstra on the backwards graph beginning at the target is performed. When the two searches 'meet', the shortest path from the source to the target is known. Figure 3.4 illustrates the different search spaces of Dijkstra and Bidirectional-Dijkstra.

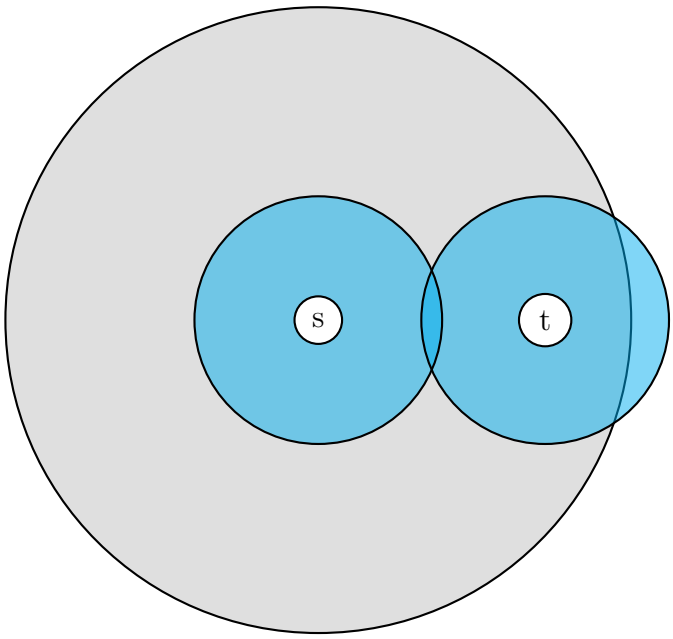


Figure 3.4: Exemplary illustration of the search spaces of Dijkstra (gray) and Bidirectional-Dijkstra (blue).

3.2.2 Contraction Hierarchies

Prior to Contraction Hierarchies, Sanders et al. [16] developed the concept of *Highway Hierarchies*. It is based on the following observation: In a road network, a hierarchy-level can be assigned to every edge. Edges corresponding to side roads in a village get a low level, whereas edges corresponding to motorways get a high level. This approach can be generalized to an arbitrary number of levels. The concept of *Contraction Hierarchies*, introduced by Geisberger et al. [6] uses this realisation. One part of their idea is to remove a node v and insert the necessary arcs, *shortcuts*, between the neighbours, such that their pairwise distances are preserved. This is called *contraction* of node v. Figure 3.5 shows an example of contracting a node.

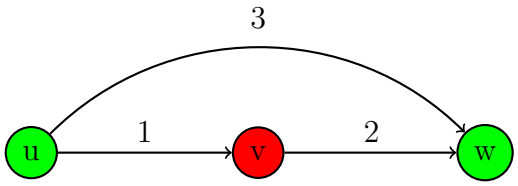


Figure 3.5: Example of contracting a node. The graph before and after the contraction of node v. Note that if the arc *(u, w)* were not existent when contracting v, this arc would be inserted as a shortcut.

All nodes are contracted in a specific order, called *contraction order*. It is chosen through the use of heuristics, with the aim of putting nodes corresponding to streets

in villages at the bottom and nodes corresponding to highways at the top of the ordering. The intention is to insert as few shortcuts as possible. One common heuristic to define the contraction order is to consider the number of shortcuts to be added, if a node would be contracted directly, minus the respective number of removed arcs. These values are called *edge differences*. The order is then defined according to increasing edge differences.

In practice, contraction of nodes does not result in deleting them from the graph. Instead, during the iterative contraction process, all previously contracted nodes are ignored.

Shortest path queries are performed on the contracted graph, using a special Bidirectional-Dijkstra. From the source node only arcs to nodes with higher ordering are considered. From the target node in the backwards graph, only arcs to nodes with higher ordering are considered. The intersection of the settled nodes originating to the source and to the target yields the minimal distance between source and target. Geisberger et al. state that even for huge networks, like the whole road network of Western Europe, query times in the order of milliseconds can be achieved. For further details, consider [6].

3.2.3 Transfer Patterns

Developed by Bast et al., *Transfer Patterns* [8] is one state-of-the-art solutions to compute optimal paths in transit networks. It makes strong use of precomputation, enabling queries to be answered in a few milliseconds. It is based on the observation that for a fixed source-target combination transfers of optimal paths occur only at few sequences of stations. These sequences are called *transfer patterns*. For instance, the fastest connections from Freiburg to Zurich might either be direct or include a transfer in Basel. Assuming the transfer patterns are precomputed, at query time only the corresponding direct connections need to be considered. In the above mentioned paper, exact and heuristic ways of precomputing the patterns are described. The heuristics allow a precomputation time linear in the network size, with only few non-optimal results. The latter is confirmed by the results of Sternisko [17]. Moreover, Bast et al. consider a multi-modal scenario. However, it is limited to transit and walking to nearby stations. As optimality criteria, duration and number of transfers are applied. Braun [13] explored the extensibility to a multi-modal scenario similar to ours. However, basic questions remain open. It is not clear how to compute concise and diverse sets of paths. Moreover, handling location-to-location queries is an unsolved problem.

3.3 Combining Road and Transit Networks

We introduce a model which combines transit, car and walking networks. We call it *combined model*. It uses existing road and transit models, augmented by the necessary connections. Simple extensions allow to incorporate further road networks.

3.3.1 Model

Assumptions, Definitions and Restrictions. The graphs of the car and walking network, G_c and G_w, are assumed to be given in the representation introduced in section 3.1.1. The transit network is assumed to be given in the time-expanded representation, the corresponding graph is called G_t.

We use the notion of *stops* to refer to the locations for which a transit or road node exists. That is, a stop can correspond to nodes in the transit and road networks. A stop for which a transit node exists is called *station*.

We restrict transitions between the networks to stations. Transfers are limited to road-network-to-transit and vice versa. This excludes (sub-)paths like "walk x minutes then take a car for y minutes". In view of the fact that the remaining (sub-)paths "walk for x' minutes" and "take a car for y' minutes" are not excluded, we deem this is reasonable.

Embedding Existing Models. For all stops induced by G_t, G_c, G_w a *start (S)* and *end (END)* node is added. A node of the car network is denoted by C, one of the walking network by W. An arc from u to v with cost c is denoted by (u, v, c). For each non-station the following arcs are added to the nodes corresponding to the stop: $(S, W, 0)$, $(S, C, 0)$, $(W, E, 0)$ and (C, E, c_{tp}), where c_{tp} denotes the transfer penalty costs. Figure 3.6 shows an example comprising these arcs.

For all stations an *exit (EX)* and an *entrance (EN)* node is added.

The nodes of the time-expanded graph, which are of type *arrival (A)*, *departure (D)* and *transfer (T)*, are augmented by the arcs: (A, EX, c_{tp}), (EN, T, c_*), where c_* denotes special costs, indicating that this arc should be treated separately. In section 3.3.2 we explain why and how. C_{cl}, W_{cl} denote the car and walking node closest to the station and cc_{cl}, wc_{cl} their respective costs. Embedding the road networks is continued by adding the arcs (EX, C_{cl}, cc_{cl}), (EX, W_{cl}, wc_{cl}), $(C_{cl}, EN, c_{tp} + cc_{cl})$, (W_{cl}, EN, wc_{cl}). The construction of the combined graph is finalised by iterating over all stations and adding arcs of cost zero from *start* to *entrance* and *exit* as well as from *entrance* and *exit* to *end*. Figure 3.7 shows an example of a station.

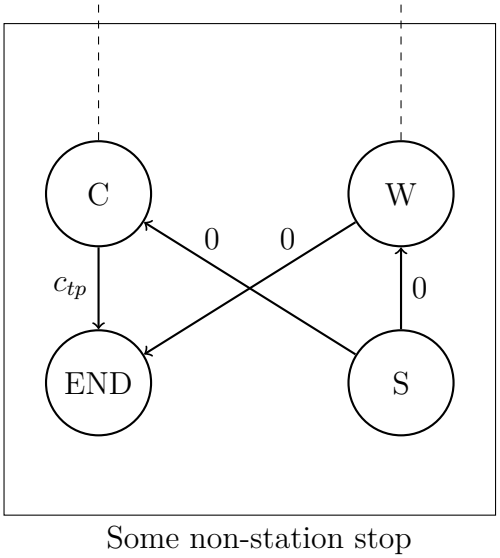


Figure 3.6: Example of a combined graph, it shows a *stop* which is not a *station*.

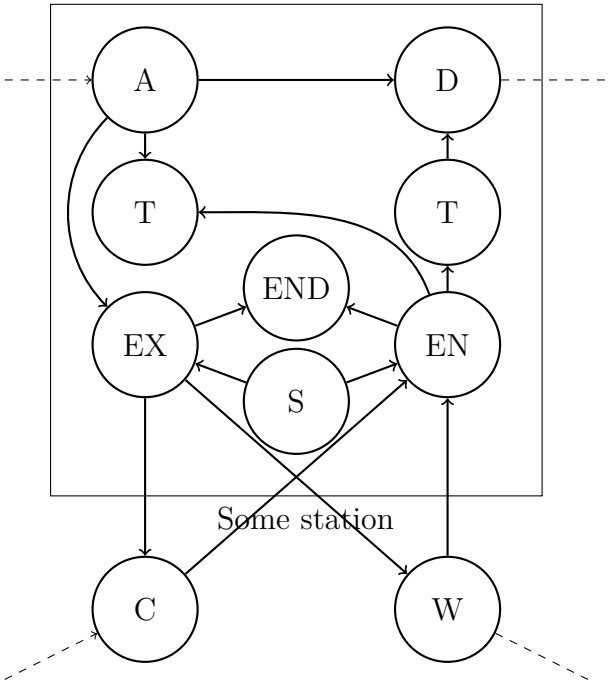


Figure 3.7: Example of a combined graph, it shows a *station*. For simplicity, arc costs were ommited.

3.3.2 Shortest Path Calculation

In our setting, queries depend on a target and source stop and the earliest possible departure time. To answer arbitrary location-to-location queries, we use the stops closest to the given locations.

Partially Contracting the Road Graphs. As explained in section 3.3.1, the combined model embeds a car and walking graph. To accelerate query computation we use a technique presented by Delling et al. [14]. The idea is to *partially contract* the road graphs before embedding them in the combined graph. The objective is to contract as many nodes as possible while keeping the set of nodes which are closest to any station, S_{cl}, uncontracted. That is, car and walking distances between all pairs of stations will be preserved. Not contracting these graphs would lead to unnecessary propagation of results when only interested in the distances. Note that this does not restrict our approach, as the paths of contracted graphs can be restored efficiently, for details, consider [6].

Each road network is contracted as follows: Using a priority queue with edge difference (see section 3.2.2 for details) for all nodes, except the ones in S_{cl} to which value infinity is assigned. The priority queue determines the order of contraction. Contraction is performed until one of the nodes in S_{cl} is popped from the queue, or the edge difference exceeds a fixed threshold (we use 10). The latter ensures that the number of inserted shortcuts stays moderate.

Performing Stop-to-Stop Queries. In the following, we assume that increasing numbers are assigned to the contracted nodes, corresponding to the contraction order. We call these numbers *contraction numbers*. We assume all non-contracted nodes (all transit, start, end, exit and entrance nodes) are assigned contraction number infinity. Furthermore, we consider the graph where only arcs to nodes with non-decreasing contraction numbers exist for each node. We refer to this as the *upwards graph*.

The computation of the shortest paths for a source stop s, target stop t and earliest departure time d can be divided into five parts:

1. Calculate the paths of walking exclusively and using the car exclusively from s to t, using Contraction Hierarchies in the respective road graphs.
2. Compute the optimal paths from s to all nodes S_{cl}, using multi-criteria Dijkstra. Consider only arcs in the upwards graph. Arcs from entrance nodes e to transfer nodes tr are only valid, if the time of tr is greater or equal to d plus the total duration of the current path to e.
3. For the car and walking network, use Dijkstra to compute the optimal path from t to all nodes S_{cl}. This is efficiently done by performing a separate query

from t in the upwards graph of the backwards graph, ignoring arcs to entrance nodes.

4. Extend the paths from s to u, where $u \in S_{cl}$, as calculated in 2., by the remaining paths from u to t, as calculated in 3.
5. Return the optimal paths of the union of those calculated in 1. and 4.

Concerning the multi-criteria cost functions we consider, for walking exclusively and using the car exclusively, only duration is relevant. Therefore, both can be calculated using Contraction Hierarchies. Note that both computations are necessary since the paths of walking exclusively and using the car exclusively determined in step 3 and 4 of the algorithm might not be optimal. In step 2, arcs from entrance to transfer nodes need to be treated in a special way, since each transfer node corresponds to a specific time, whereas entrance nodes are time-independent. Recall that we use a priority queue order which pops labels by increasing duration. Here we can use this property and sort outgoing arcs of entrance nodes by increasing time. This allows to efficiently keep track of the potentially valid arcs.

For simplicity, queries begin at the start node of s and end at the end node of t. In the following section, we explain why and how start and end nodes can be removed.

3.3.3 Improving Compactness of the Model

To make the combined model more compact, start and end nodes and their corresponding edges can be removed.

The shortest path calculation has to be slightly adjusted. In step 2 of the algorithm, we have to distinguish the case of beginning the multi-criteria Dijkstra at a station or a normal stop. If it starts at a station, Dijkstra has to begin at the stations entrance and exit node. If it begins at a normal stop, Dijkstra has to begin at the stops car and walking node. To get the optimal paths of the target, we also distinguish the case of it being a station and a normal stop. If it is a station, the optimal paths are the optimal union of the paths ending at the stations entrance and exit node. If it is a normal stop, the optimal paths are the optimal union of the paths ending at the stops car and walking node. If the transfer penalty is not zero, it needs to be added to all optimal paths ending at the car node. This finalises the adjustment of the shortest path calculation. Its correctness can simply be seen by comparing an imaginary shortest path computation on the normal and the compact model.

The number of stops is usually very large (two million for our dataset of New York City). Hence, shortest path computation becomes more efficient, since unnecessary propagation of optimal labels is avoided. Especially for multi-criteria functions, this is an important aspect. Moreover, the memory usage of the graph is reduced.

4 Multi-Criteria Shortest Paths

We discuss different optimality criteria and how to combine them to get concise and diverse sets of optimal paths. In the previous sections we did not use a specific definition for the optimality of paths. Here we catch up on this.

In road networks, exactly one criterion, i.e. *duration*, is sufficient in many cases. In transit networks, convenience is another criterion [8]. Some people would want the fastest connection, while others would prefer the most convenient one. We measure convenience by the number of transfers (the number of boarded vehicles). We call this *transfer penalty*. To optimise multiple criteria in a non-dominating way we use Pareto Sets.

4.1 Pareto Sets

To model n criteria, n-tuples are used. Each component consists of a non-negative entry, representing one criterion. Sometimes we refer to these tuples as *costs*. For example, when modelling *(duration, transfer penalty)*, the tuple $(100, 2)$ corresponds to "duration is 100 minutes and one needs two transfers".

We briefly recapitulate the definition of *Pareto Sets*. For two n-tuples t_1, t_2: if $t_1 \leq t_2$ using component-wise comparison, we say that t_1 *dominates* t_2. Let S be a set of n-tuples. The Pareto Set $P \subseteq S$ is defined such that for all $s \in S : \exists p \in P$ with $p \leq s$ and for $p_1, p_2 \in P$ with $p_1 \neq p_2$ and p_1 does not dominate p_2. For instance, the Pareto Set of $\{(100, 2), (105, 2), (130, 1)\}$ is $\{(100, 2), (130, 1)\}$. This reflects that $(100, 2)$ dominates $(105, 2)$, and that $(130, 1)$ and $(100, 2)$ are incomparable.

With Pareto Sets, multiple tuples can be optimal at a specific node. Each tuple is included in a label. Moreover, a label corresponds to a path from the source to the labels node.

4.2 Criteria

Determining a set of criteria that leads to concise and diverse sets of paths is a challenge in itself. We discuss several combinations of criteria and demonstrate their usefulness with respect to the above mentioned properties.

4.2.1 Duration and Transfer Penalty

Using *duration* and *transfer penalty* as Pareto criteria seems reasonable at first sight. The transfer penalty can be extended to take car transfers into account, by incrementing it not only when boarding a transit vehicle, but also when boarding a car. However, the results are not satisfying. In our multi-modal scenario, the Pareto Set of optimal costs usually has the following form:

$$\{(x, 1), (y, 0)\} \text{ with } x < y.$$

The tuple $(x, 1)$ corresponds to "taking a car for the whole way". The remaining tuple corresponds to "walking the whole way". Paths including the usage of the transit network are completely missing. The reason for these one-sided results is that on the one hand, using a car is usually the fastest solution and it requires only one transfer. On the other hand, walking exclusively has transfer penalty zero, hence it is always optimal in the Pareto sense. The former is aggravated by the fact that our model contains neither turn restrictions, traffic lights nor further traffic information. We call this the **blue light effect**, since it reminds of the rules applying to police cars.

4.2.2 Duration, Transfer Penalty and Car Duration

To generate more diverse results, we use *car duration* as an additional criterion. Diversity increases, but the number of optimal paths grows enormously. Moreover, many very similar paths are optimal. That is, we do not consider the set of optimal paths to be concise. It brings along two problems. Firstly, the results are too numerous to be considered practical to show to a human. Secondly, query times become infeasible. Figure 4.1 shows an excerpt of an example query.

Obviously, some of the results are very similar, although all of them are optimal in the Pareto sense. Two fundamentally different approaches can be considered in order to solve this problem: On the one hand one could try to prohibit the generation of similar solutions by restricting the model, on the other hand one could filter the Pareto Set. The work presented in [13] focuses on the former approach and indeed a significant reduction of computation time can be achieved, however, at substantial expense of optimality. In this thesis, we focus on the latter approach. In the following we study several ideas how to filter Pareto Sets of the three above mentioned criteria.

4.3 Filtering Pareto Sets

In this section we focus on filtering Pareto Sets to concise and diverse subsets.

duration	transfer penalty	car duration
0:28:57	1	0:28:57
	...	
1:43:43	3	0:16:35
1:44:01	3	0:16:26
1:44:09	3	0:16:04
1:44:34	5	0:11:07
1:44:36	3	0:15:56
1:45:12	4	0:15:51
	...	
7:06:00	0	0

Figure 4.1: Excerpt of the optimal costs of an example query using duration, transfer penalty and car duration as Pareto criteria.

4.3.1 Discretisation

As figure 4.1 shows, some of the Pareto optimal results are very similar. A first approach is to *round* the durations from exactness to the second to full minutes. We argue, that in practice, humans would not distinguish these solutions. For the above example, this already reduces the number of Pareto optimal results, as figure 4.2 shows.

duration	transfer penalty	car duration
0:29:00	1	0:29:00
	...	
1:44:00	3	0:17:00
1:45:00	3	0:17:00
1:45:00	3	0:17:00
1:45:00	5	0:12:00
1:45:00	3	0:16:00
1:46:00	4	0:16:00
	...	
7:06:00	0	0

Figure 4.2: Excerpt of the optimal tuples of an example query. Durations were rounded to full minutes. Green tuples are still Pareto optimal after rounding, gray ones are not.

An extension of the above rounding to minutes, is to *discretise* car duration to certain blocks (for example, 10 minutes). This idea was also introduced in [14]. However, it was used as a heuristic during query time to reduce computation complexity. Our

intention is to obtain a coarser, smaller, yet still representative Pareto Set. The result for the above example is shown in figure 4.3.

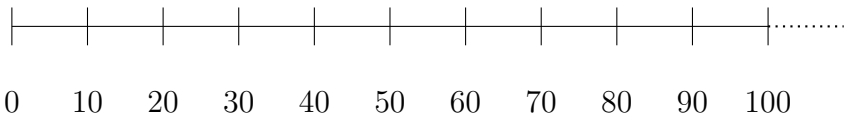
duration	transfer penalty	car duration	discretised car duration
0:29:00	1	0:29:00	0:30:00
		...	
1:44:00	3	0:17:00	0:20:00
1:45:00	5	0:12:00	0:20:00
1:45:00	3	0:16:00	0:20:00
		...	
7:06:00	0	0	0

Figure 4.3: Excerpt of the optimal tuples of an example query, before and after discretisation at blocks of 10 minutes. Green tuples are still Pareto optimal after the discretisation, gray ones are not.

While the remaining tuples can be called representative in terms of the original Pareto set, the following issues still exist. Discretisation might lead to pairs of costs which are very similar. This is the case, if two costs at the common 'border' of two consecutive blocks are chosen as representatives. Consider the following example:

duration	transfer penalty	car duration	discretised car duration
1:40:00	3	0:21:00	0:30:00
1:43:00	3	0:19:00	0:20:00

After discretising car duration to blocks of 10 minutes, both tuples remain Pareto optimal. A solution is to halve the block size and choose the representative of every second block. Considering the border problem solved, the filtered results are still not satisfactory. Using a fixed block size implies the same level of absolute coarseness for all car durations. The following image illustrates this:



We argue that in practice, a user does not want the same level of absolute coarseness, but an increasing one. This can be achieved by using increasing block sizes. For instance, setting the upper border of block i to $10 \cdot 2^i$ minutes yields the following block distribution:

Note that due to Pareto optimality, the duration of the optimal costs using exclusively the car from source to target is an upper bound for the car duration of all other Pareto optimal costs. We call this bound *pure car duration*. While the number of optimal labels decreases (compared to fixed block sizes), it is, especially for queries with large pure car duration, still too large for practical usage. Moreover, we still consider some of the remaining tuples as not desirable. In the following section, we introduce a filtering method which solves these problems.

4.3.2 Types and Thresholds

Discretisation as described above is based on the intuition that many Pareto optimal solutions are similar, requiring to choose a representative subset. Each solution itself is considered reasonable. Examining the solutions more closely indicates that the latter is not the case. Consider the following example:

"Walk 120 minutes, then ride the train for 10 minutes and finally take a car for 5 minutes to arrive at the target"

While this path might be Pareto optimal, we consider it unlikely that anyone will favour it. Especially if a path like *"Take the car for 15 minutes to arrive at the target"* exists as well. The idea underlying this example is: if a path already requires much walking, one will not take a car for just a few minutes. Recall that we did not consider walking duration as an explicit Pareto criterion. We discuss this issue in the next section. In the following, we analyse transit, car and walking duration with respect to their *relative durations (RD)*. We classify them as reasonable and unreasonable combinations. Based on this, we determine three **types** of paths, incorporating all combinations classified as reasonable.

As relative durations we use *zero (z), little (l), much (m)*. Hence $RD := \{z, l, m\}$. We call a set of RD's *valid* if at least one element is m. Our classification is based on the following observations:

- Cars are available everywhere and fast, but expensive.
- Transit is medium-fast and medium-expensive, but at limited availability.
- Walking is possible everywhere and cheap, but slow.

Figure 4.4 contains all combinations of RD's, annotated according to our classification as *reasonable (✓)* or *unreasonable (✗)*.

We assume the obvious order of $z < l < m$. The classification is consistent to the effect that for each triple classified as reasonable each valid component-wise smaller triple is classified reasonable, too. This can easily be inferred from the above mentioned table.

transit duration	walking duration	car duration	classification
z	z	z	✗
z	z	l	✗
z	z	m	✓
z	l	z	✗
z	l	l	✗
z	l	m	✗
z	m	z	✓
z	m	l	✗
z	m	m	✗
l	z	z	✗
l	z	l	✗
l	z	m	✗
l	l	z	✗
l	l	l	✗
l	l	m	✗
l	m	z	✓
l	m	l	✗
l	m	m	✗
m	z	z	✓
m	z	l	✓
m	z	m	✗
m	l	z	✓
m	l	l	✓
m	l	m	✗
m	m	z	✓
m	m	l	✗
m	m	m	✗

Figure 4.4: All combinations of RD triples for transit, car and walking duration with our classifications. White background indicates that an RD triple is not valid. Gray color indicates that a triple is prohibited due to the graph model. From the remaining triples, the ones classified reasonable are green, the others are blue.

Finally, we determine three **types** of all reasonable solutions:

1. Only car.
2. Much walking, much transit, no car.
3. Much transit, little walking, little car.

Here, the attributes *much* and *little* should include all smaller relative durations. The types are complete to the effect that exactly all elements classified as reasonable are included. This can again be deduced simply from the table in figure 4.4.

For practical purposes, *much* and *little* need to be defined concrete. Incorporating that *little* depends on the type of transportation, we chose the following definitions. All durations are given in minutes and we omit the unit *minutes* for all function values:

- $zero(*) := 0$
- $little(\text{walking}) := 10$
- $little(\text{car}) := 0$ if pure car duration < 20, $max(10, 25\% * \text{pure car duration})$ otherwise
- $much(*) := \infty$

These definitions provide the necessary **thresholds** for all considered durations.

Actual filtering consists of two steps. Firstly, all labels not belonging to any type are discarded. Secondly, with the following function, all car durations are transformed to their relative durations:

$$rd(\text{car duration}) = \begin{cases} 0, & \text{if car duration} = 0 \\ 1, & \text{if } 0 < \text{car duration} \leq \text{little(car)} \\ 2, & \text{if little(car)} < \text{car duration} < \text{much(car)} \end{cases}$$

The Pareto Set of the transformed triples constitutes the result. We call the whole concept **Types aNd Thresholds (TNT)**. Figure 4.5 shows the final result of a query performed on real data.

4.3.2.1 How to Take Walking Duration into Account?

In the previous section we proposed the concept of TNT, for which walking duration is an important element. We discuss two ways of calculating it.

When filtering a Pareto Set of triples with criteria (duration, transfer penalty, car duration), the **implicit walking duration** can be accumulated in a hidden variable for each triple. However, this can lead to suboptimal results. The example in figure 4.6 illustrates how solutions can get lost, if walking duration is not considered as an explicit Pareto criterion.

The problem of suboptimal paths can be alleviated by discarding certain labels during query computation. As soon as a label does not belong to a certain type, its

duration	transfer penalty	car duration	path-summary
0:23:00	1	0:23:00	23C
1:12:00	3	0:10:00	8W-4-23T-1W-2-26T-10C
1:47:00	2	0:10:00	10W-19-69T-10C
1:48:00	3	0	8W-4-23T-5W-1-19T-1-19T-30W
2:05:00	2	0	10W-19-24T-2W-6-34T-31W
2:35:00	1	0	75W-8-42T-31W
4:46:00	0	0	286W

Figure 4.5: Filtered results of a random query performed on real data. Paths are summarised with the following regular expression:
(*duration means-of-transportation* (- *waiting-time*)?)*
Means of transportation is either *car (C)*, *walking (W)* or *transit (T)*. Duration and waiting time are rounded and given in minutes.

extensions also cannot belong to that type. Therefore, if a label solely belongs to type 3 and an extension does not belong to type 3, the extension can be discarded. This implies, that an extension of a label belonging only to type 3, which exceeds the walking duration threshold, can be discarded. Figure 4.7 shows an example which illustrates why this can indeed improve the quality of the final results. Moreover, discarding labels during query computation reduces the computation time. We discuss this further in section 5.3.

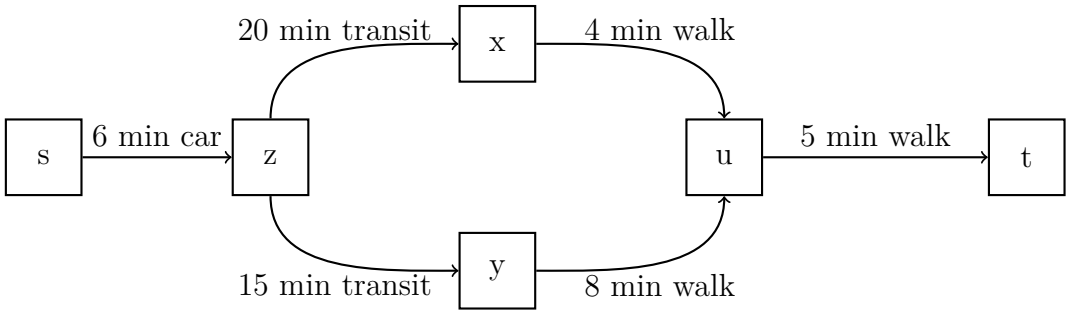


Figure 4.6: Example for the suboptimality of using implicit walking duration with Pareto criteria duration, transfer penalty, and car duration. Imagine a query from stop s to stop t: at u the path via y would be optimal and dominate the path via x. Hence, the path via x would be discarded. At t, the path via y exceeds the threshold *little(walking)*, hence it is discarded during filtering. This is suboptimal, because at t the path via x does not exceed the threshold *little(walking)*.

An optimal solution is to use **walking duration as a separate Pareto criterion**. However, this increases computation time. Moreover, it calls for additional post-processing. Otherwise, many optimal paths would be similar, varying in the amount of walking duration. Therefore, we post process the results by returning

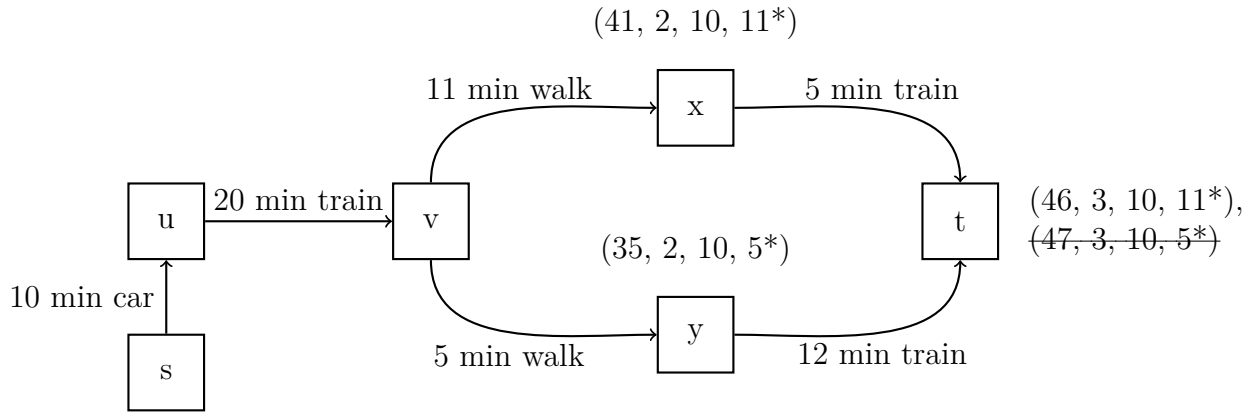


Figure 4.7: Example why not discarding labels without type during query computation can affect the final result. The nodes x, y, t are annotated with their optimal labels (duration, transfer penalty, car duration, implicit walking duration). The asterisk for implicit walking duration indicates that it is not used as a Pareto criterion. Consider a query from s to t. Then, at t, the label stemming from y, l_y, would be dominated by the one stemming from x, l_x. Hence, l_x would be kept. However, since it does not belong to any type, it would be discarded in the post-processing step. Therefore, l_x can be discarded during query computation. Then l_y is not dominated and also survives the post-processing step.

the Pareto Set with respect to duration, transfer penalty, and car duration. That is, after ensuring that no desired optimal solution got lost, we filter back to the original Pareto criteria. This keeps at most one solution per triple (duration, transfer penalty, relative car duration). However, computation times are expected to increase even more. The experimental results confirm this, for details see section 6.

In the following section, we discuss different speed-up techniques to get a grip on this problem.

5 Exploring Speed-up Approaches

5.1 General

5.1.1 Rounding on Transfers

In section 4.3.1, we introduced the idea of rounding to full minutes. A special form can also be applied during query computation, yet preserve optimality in a certain sense. In our implementation, the durations of arc costs are given exactly to the second. This accuracy is necessary to avoid accumulating error (compared to using full minutes). For road networks, we calculate durations depending on distances of nodes. The GTFS data [18], which we use to model the transit network, provides durations in seconds. Therefore, this is indeed practically relevant. We assume that no kind of public transportation in practice provides accuracy by seconds. The speed of humans in terms of walking and car usage varies too. Thus, we propose to **round up durations, immediately before transfers, to full minutes**. This can also be interpreted as a coarse transfer buffer. With respect to reality, we consider this an optimality preserving technique. Figure 4.1 and 4.2 illustrate the rounding effect on an example based on real data. We expect rounding during query time to noticeably speed-up computation time. Arc relaxations will happen for less labels. Moreover, the Pareto Set of labels attached to each node will shrink, resulting in less comparisons when a new label is inserted.

5.1.2 Flattening the Transit Graph

Originally, our intention was to evaluate a heuristic to speed-up the computation of transfer patterns for multi-modal networks. At that stage of the thesis, we were not aware that finding an appropriate cost function would be a severe problem. Therefore, we first focused on a heuristic approach of making the transit graph independent of time. We call this *flattening* the transit graph. The aim was to make transfer pattern computation significantly faster, taking into account a fraction of non-optimal results. Trips which do not overtake each other and have the same stop sequences are grouped to *lines*. For the transit graph, exactly one trip per line is kept as a representative. The heuristic then diversifies into two branches. In the first branch, it is encoded in the graph that the representatives are reachable immediately after arriving at a station. We call this the *immediate heuristic*. In

the second branch, the representatives are reachable after the average difference of the subsequent departure times of the respective lines. We call this the *line average heuristic*.

To determine if the heuristics fulfil the assumption of significantly improving computation times, we evaluated them for location-to-location queries. As Pareto criteria we chose duration, transfer penalty, and car duration. Since computation times were not significantly lower, we rejected this approach. For details of the evaluation, consider section 6.2.1.

5.2 Discretising during Query Computation

In section 4.3.1, we introduced the idea of discretising Pareto criteria in a post-processing step, to filter the numerous optimal labels. Recall that incorporating car duration as a separate criterion heavily increases query computation time. To reduce it, we propose the heuristic of performing **discretisation during query computation**:

- Pareto criteria are duration, transfer penalty, and discretised car duration.
- Car duration is a hidden parameter of each label. It is used to correctly calculate the corresponding discretised car duration.
- Tie-breaking is necessary to decide for labels with equal Pareto criteria, as to which one to keep. A natural decision is to choose the one with less car duration.
- To break ties correctly, the priority queue order is chosen to compare the criteria in the order duration, transfer penalty, car duration (not discretised car duration).

Compared to performing discretisation as post-processing, performing it during query computation does not necessarily preserve optimality. Figure 5.1 illustrates this with an example.

Since it is not obvious that each settled label corresponds to a path, we show this by proving that each settled label can never be replaced by another one.

Claim: *For discretisation during query computation, a settled label can never be replaced by another one.*

Proof: Assume a label l_1 exists which is settled and for the same node a label l_2 exists which afterwards is popped from the priority queue and has equal Pareto criteria. The order of the priority queue implies $l_1.\text{car-duration} \leq l_2.\text{car-duration}$. □

For each block, we call the representative obtained by applying discretisation as a post-processing step *optimal representative*. Performing discretisation during query

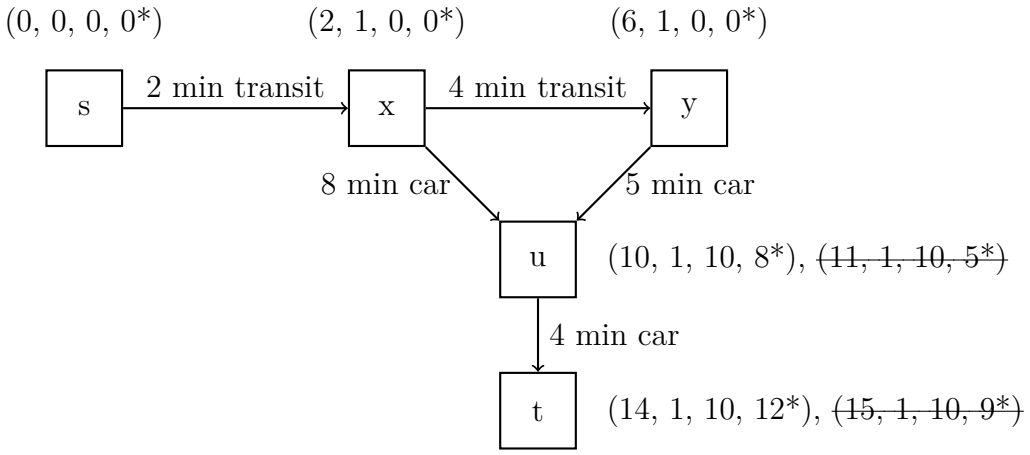


Figure 5.1: Example for the non-optimality of discretising during query computation. An asterisk is used to tag the hidden parameter car duration. Assuming a block size of 10 minutes, the label (15, 1, 10, 9*) at stop t would be missing. It would be available, if discretisation were performed as a post-processing step. For simplicity, transfer times were neglected.

computation yields an average recall of the optimal representatives of around 90%. For details of the experimental results see section 6.

5.3 Types and Thresholds

5.3.1 Pruning with Pure Car and Walking Duration

The types of paths and their thresholds introduced in section 4.3.2 can be used to discard labels during query computation while preserving optimality. As soon as a label belongs to no type, it cannot belong to the filtered Pareto Set, hence it is superfluous and can be discarded.

Recall, part of the shortest path algorithm introduced in section 3.3.2 are Contraction Hierarchies to compute the shortest paths of exclusive walking and exclusive car usage. Recall, the duration of the latter was defined as *pure car duration*. Accordingly, we call the duration of the former *pure walking duration*. Since both are computed very fast (in the order of milliseconds) we can use them for our pruning purposes.

That is, after step 1 of the shortest path algorithm, pure car duration is known. Therefore, in steps 2-4, only labels belonging to type 2 or 3 need to be kept. This induces the following pruning rule for *all* labels:

Pruning rule: *A label can be discarded if:*

1. *duration > pure walking duration* or
2. *walking duration > little(walking) and car duration > 0* or
3. *car duration > little(car)*

Claim: *The pruning rule preserves optimality.*

Proof: Given a label, for extensions of the label, all Pareto criteria can only increase. Thus, a label dominates its extensions. Any label which fulfils 1., will be dominated by the label corresponding to pure walking duration, which has the form (pure walking duration, 0, 0). Any label which fulfils 2. can not belong to type 2 or 3. The same applies to 3. □

5.3.2 Implicit Walking Duration

As mentioned above, to filter labels to the three types, their walking durations have to be available. We already showed that not using walking duration as an explicit Pareto criterion can lead to non-optimal results. Nevertheless, in practice we expect the difference to the optimal results to be minor.

To efficiently keep track of the implicit walking duration, we propose a similar approach as for discretising during query computation. The walking duration should be kept in a hidden parameter of the label. For labels with equal Pareto criteria, the label with less walking duration should be kept.

It is noteworthy that using implicit walking duration will not affect the quality of type 2 paths (much transit, much walking, no car). For the correctness of this argument, consider the following proof.

Claim: *Let (A) be the result of using the Pareto criteria duration, transfer penalty, car duration, and walking duration during query computation. Let (B) be the result of using Pareto criteria duration, transfer penalty, and car duration with implicit walking duration during query computation. We compare two labels by their common Pareto criteria.*
For a fixed node, for each label l of type 2 from (A), a label l' of the same node of type 2 from (B) with $l' \leq l$ exists.

Proof: Each l of (A) corresponds to a path $l_{u_1}, \ldots, l_{u_n} = l$, where l_{u_i} at node u_i denotes the predecessor of $l_{u_{i+1}}$. We prove by induction on that path, that for each l_{u_i} of type 2, exists l'_{u_i} of type 2 produced by (B) with $l'_{u_i} \leq l_{u_i}$. For u_1 this is obvious. Assuming there exists l'_{u_i} with $l'_{u_i} \leq l_{u_i}$, we show there exists $l'_{u_{i+1}} \leq l_{u_{i+1}}$: Since the same graph is used for (A) and (B) and there must exist an arc from u_i to u_{i+1} with costs $c = l_{u_{i+1}} - l_{u_i}$, we have $l'_{u_i} + c \leq l_{u_i} + c = l_{u_{i+1}}$. Therefore, and due to Pareto optimality where labels can only get dominated by labels with lower costs, in (B) at node u_{i+1} exists $l'_{u_{i+1}}$ with $l'_{u_{i+1}} \leq l_{u_{i+1}}$. □

6 Experiments

In this section, we evaluate the heuristics and filtering methods. We briefly evaluate the heuristics of flattening the transit graph. We compare discretisation and using types and thresholds with their heuristics. We measure query computation duration and quality.

6.1 Setup

Our implementation of combined graph, as described in section 3.3.2, is written in `C++` and compiled with `GCC 4.6.3` with the `-O3` flag. The experiments were performed on a machine with 96GB of RAM and two Intel Xenon E5649 CPUs with 8 cores, each having a frequency of 2.53 GHz (exactly one core was used at a time). The used OS is Ubuntu `12.04`, operating in 64-bit mode.

To instantiate the multi-modal networks we used publicly available OSM [19] and GTFS data [18]. We only used data of a randomly selected Monday. OSM data was chosen to cover the terrain corresponding to the GTFS data. For walking, we assumed an average speed of 4 km/h. For the car network, average velocity was chosen depending on the road type, ranging from 5 to 110 km/h. We evaluate our algorithms on the networks of *Austin, Dallas, Toronto* and *New York City*. In the following, the latter is abbreviated with *New York*. The table in figure 6.1 contains an overview over the most important properties of these networks. The road network graphs are symmetric and filtered to their largest connected component. Due to time restrictions, we did not use the most memory-efficient solution. Start nodes and their corresponding arcs were not removed. End nodes were not removed, however we deleted their corresponding arcs. Compared to a more space-efficient solution described in 3.3.3, the influence on computation time is negligible. Furthermore, rounding on transfers is used for all algorithms.

For each multi-modal network, the following experiments were performed using 100 queries from random stops with some station within at most 1 km point-to-point distance. We chose this restriction to exclude that a significant amount of queries would be performed in areas where transit is not available. Departures times were chosen randomly from the range of 06:00 a.m. to 10:00 p.m.

	Austin	Dallas	Toronto	New York
All				
#Stations	2.7K	11.6K	10.9K	15.8K
#Stops	0.3M	1.4M	0.4M	2.0M
#Nodes	2.0M	7.5M	7.5M	16.9M
#Arcs	5.0M	18.9M	17.5M	41.1M
Transit				
#Nodes	0.6M	1.8M	5.9M	9.0M
#Arcs	1.1M	3.2M	10.3M	15.6M
Car				
#Nodes	0.3M	1.3M	0.4M	2.0M
#Nodes (core)	3.3K	20.8K	10.9K	28.0K
#Arcs	1.4M	5.9M	1.7M	8.5M
Walk				
#Nodes	0.3M	1.4M	0.4M	2.0M
#Nodes (core)	3.7K	20.8K	12.5K	32.6K
#Arcs	1.4M	6.0M	1.7M	8.6M

Figure 6.1: Overview over the most important properties of the evaluated networks.

6.2 Results

6.2.1 Flattened Transit Graph

We performed only few experiments for the heuristics of flattening the transit graph, because the expectation of significantly reduced query times was not fulfilled. The table in figure 6.2 shows the results for using Pareto criteria duration, transfer penalty, and car duration.

Note that each label corresponds to a pattern. Compared to the original transit graph, the number of found patterns increased significantly for both of the heuristics. Since the number of nodes and the number of labels per node have strong influence on the query durations, this can lead to higher running times of the heuristics. The intention behind flattening the transit graph was to reduce query computation by paying with lower quality. Since the former is not the case, we refrained from examining this approach in more detail.

6.2.2 Discretisation

In this section we compare discretisation as a post-processing method and using it as a heuristic during query computation. The former uses the Pareto criteria duration, transfer penalty, and car duration, the latter duration, transfer penalty and discretised car duration, see section 5.2 for details. As block size we chose 10

	Austin	Dallas	Toronto	New York
Original Transitgraph				
Duration (avg/50/90/99)	4/4/7/10	44/43/63/75	86/84/118/151	154/146/207/372
#Patterns (avg/50/90/99)	16/14/30/40	22/20/38/50	22/20/35/51	26/25/42/71
Flattened Transitgraph				
#Transit nodes	17K	61K	92K	134K
Immediate Heuristic				
Duration (avg/50/90/99)	3/2/4/5	41/41/57/85	24/21/37/66	57/56/85/112
#Patterns (avg/50/90/99)	21/19/36/58	30/27/50/72	27/24/42/79	33/31/59/85
Recall (avg/50/90/99)	.3/.3/.6/1	.3/.2/.5/.8	.3/.2/.4/.8	.3/.3/.5/.7
Precision (avg/50/90/99)	.2/.2/.4/.6	.2/.2/.3/.7	.2/.2/.3/.8	.2/.2/.4/.7
Line Average Heuristic				
Duration (avg/50/90/99)	2/2/4/6	42/42/61/88	28/24/46/62	56/54/80/101
#Patterns (avg/50/90/99)	20/17/38/63	28/25/48/73	27/25/43/68	33/29/58/105
Recall (avg/50/90/99)	.3/.3/.6/1	.3/.3/.5/.8	.3/.2/.5/.8	.3/.3/.5/.6
Precision (avg/50/90/99)	.2/.2/.4/.7	.2/.2/.4/.8	.2/.2/.4/.7	.3/.2/.4/.6

Figure 6.2: Flattening the transit graph: Recall and precision are given with respect to the optimal representatives. Durations are given in seconds. The table provides average, 50%-ile, 90%-ile and 99%-ile values.

minutes. We examine query duration and quality. The latter is measured in recall and precision of the optimal representatives (section 4.3.1). Additionally we use an approximate recall measure, allowing for a deviation of x minutes and y percent of the optimal solution. We call this measure *recall-x,y*. We use *recall-5,5* and *recall-10,10*. Moreover, we measure the percentage of paths for which no approximate match (with the same discretised car duration) was found.

We consider these measures necessary, since the recall itself is not convincing to determine the quality of the results. For example, assume the recall is 50%, and distinguish two cases of duration deviation of optimal representatives: 1) 0.1% and 2) 200%. Despite having equal recall values, we consider the objective loss of quality in the former case to be significantly lower than in the latter case. Figure 6.3 shows the experimental results with respect to computation duration and number of found optimal paths. Figure 6.4 reflects the quality of the found paths using boxplots.

To summarise, the heuristics lead to an improvement in duration of around factor 4. Absolute running times are roughly in the order of tens of seconds, which is too high for practical application. Moreover, the number of filtered paths is roughly around 8, which is on the border of feasibility. Precision and recall of the heuristic are both around 90%, although some paths are not even matched approximately.

Data	Algo	Duration avg/50/90/99	#Paths avg/50/90/99	#Filtered Paths avg/50/90/99
Austin	PP	4/4/6/8	16/15/26/38	6/6/9/14
	DCQ	1/1/1/1	-	6/6/9/15
Dallas	PP	43/40/60/83	24/20/40/53	8/8/11/14
	DCQ	7/7/8/9	-	7/7/10/13
Toronto	PP	86/83/114/139	24/22/39/46	7/7/10/12
	DCQ	22/22/25/26	-	7/7/10/12
New York	PP	181/173/251/418	24/22/40/57	8/8/12/14
	DCQ	37/37/46/54	-	8/8/11/14

Figure 6.3: Discretisation: Comparison of the post-processing method (PP) and as a heuristic during query computation (DCQ). Durations are given in seconds. The table provides average, 50%-ile, 90%-ile and 99%-ile values.

6.2.3 Types and Thresholds

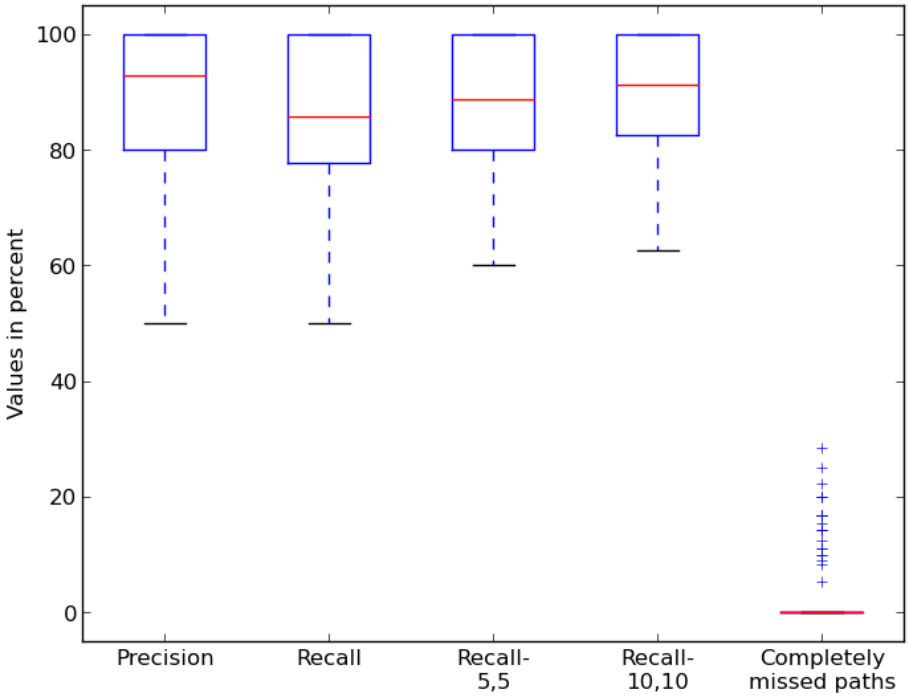
In this section, we examine the approach of using types and thresholds (TNT) with the respective speed-up techniques (section 5.3). We compare the basic algorithm with the heuristic of using implicit walking duration (IWD). For the basic algorithm, we also measure the effect of the pruning rule. We run the IWD heuristic always with the pruning rule, since it affects the quality of the results. Recall that for the basic algorithm this is not the case. Figure 6.5 shows the experimental results with respect to computation duration and number of found optimal paths.

Quality is measured in the same manner as for discretisation. Figure 6.6 shows the results. Since paths of type 3 are the only ones affected by the IWD heuristic, we additionally measure the quality of the type 3 results. That is, we separately take into account only those queries and paths, where the basic algorithm leads to optimal paths of type 3. The results are shown in figure 6.7.

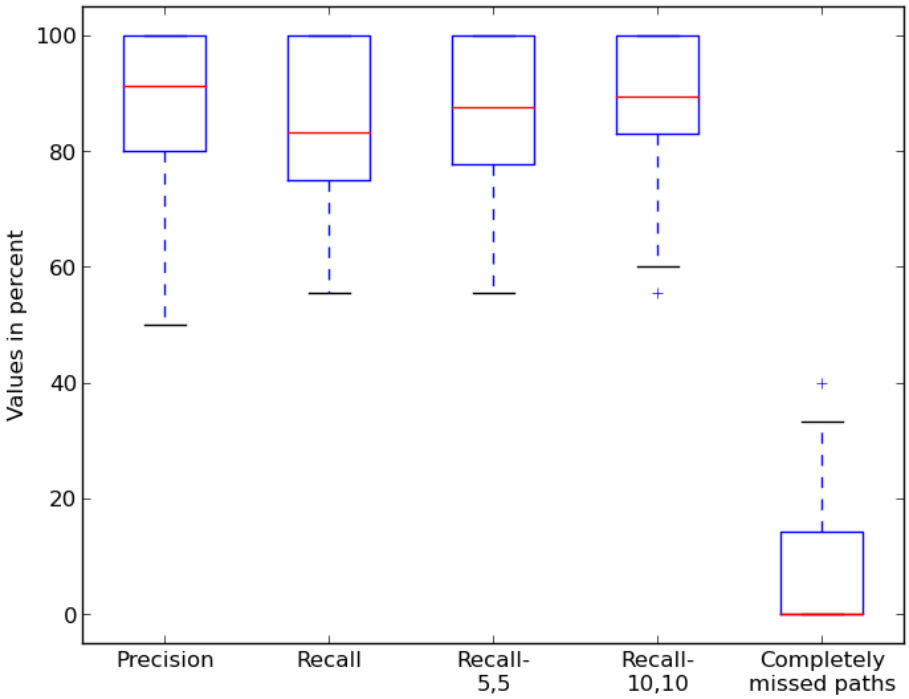
Especially for practical use, the total number of found paths and their distribution to the three types is of interest. Figure 6.8 shows boxplots for the respective data.

The experimental results show that using TNT with the basic algorithm leads to infeasible query times (order of tens of minutes for larger datasets). The pruned version runs in the order of many seconds to few minutes. The heuristic of using implicit walking duration leads to running times of a few seconds for small and medium sized datasets and to tens of seconds for larger datasets. Precision and recall are both close to 100%, although for a fraction of queries some optimal paths are completely missed. The same is true when considering only those queries and paths where the heuristic can have a negative effect on the quality. That is, the heuristic returns almost optimal results.

The number of filtered paths is for all datasets roughly around five. There is always one path of type 1 (using car exclusively), around four paths of type 2 (transit and

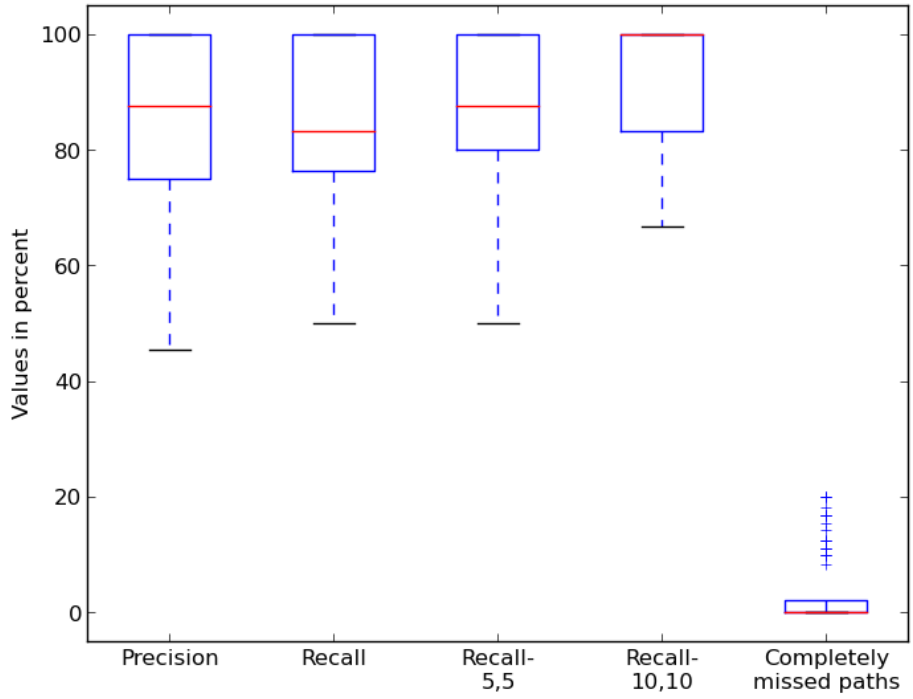


(a) Austin

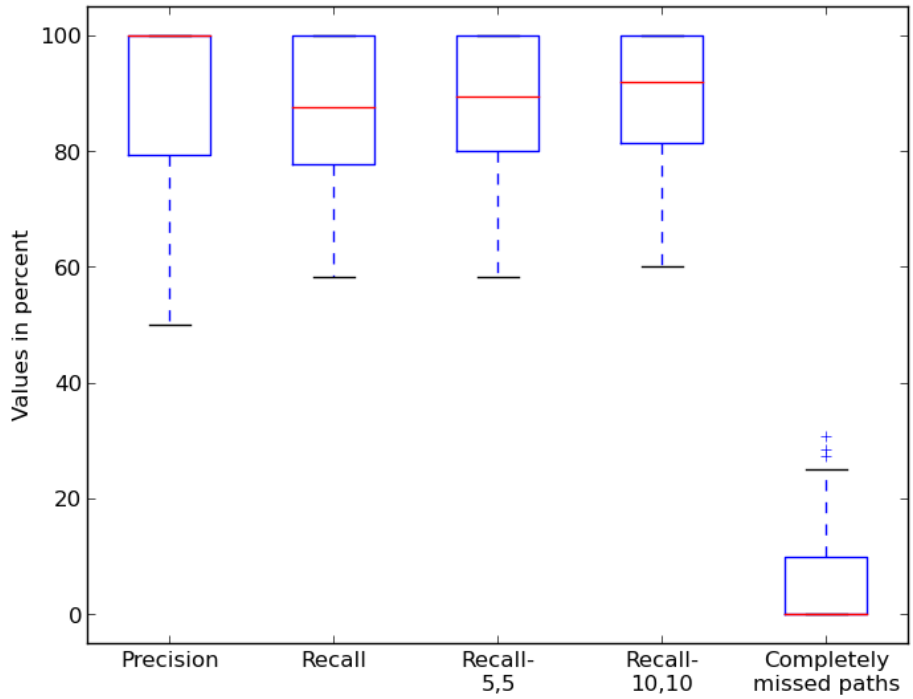


(b) Dallas

Figure 6.4: Discretisation: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets.



(c) Toronto



(d) New York

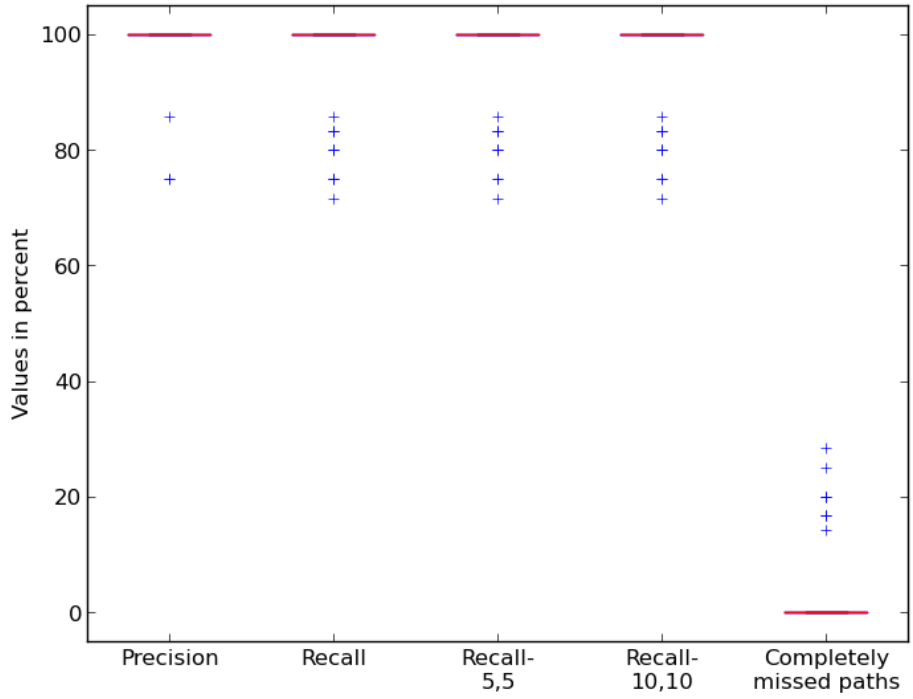
Figure 6.4: Discretisation: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets.

Data	Algo	Duration avg/50/90/99	#Paths avg/50/90/99	#Filtered Paths avg/50/90/99
Austin	Basic	22.9/19.8/35.2/60.8	61/54/109/198	-
	Basic-pruned	2.7/0.8/7.6/14.9	15/10/36/64	4/4/6/8
	IWD-pruned	0.5/0.3/1.2/2.4	5/4/7/18	4/4/6/8
Dallas	Basic	276.0/252.0/449.0/582.0	70/52/135/227	-
	Basic-pruned	24.6/25.3/53.5/84.8	21/19/42/85	5/6/7/9
	IWD-pruned	4.2/4.5/7.9/9.5	6/6/11/12	5/5/7/9
Toronto	Basic	924.0/846.0/1380.0/1620.0	123/110/215/320	-
	Basic-pruned	124.0/126.0/240.0/349.0	44/45/81/105	6/6/8/9
	IWD-pruned	12.1/13.5/21.1/24.8	8/8/12/16	6/6/8/9
New York	Basic	1770.0/1370.0/3220.0/5630.0	123/107/250/388	-
	Basic-pruned	308.0/260.0/628.0/1450.0	40/29/85/109	5/5/8/9
	IWD-pruned	54.1/25.8/81.0/298.0	7/5/11/16	5/5/7/9

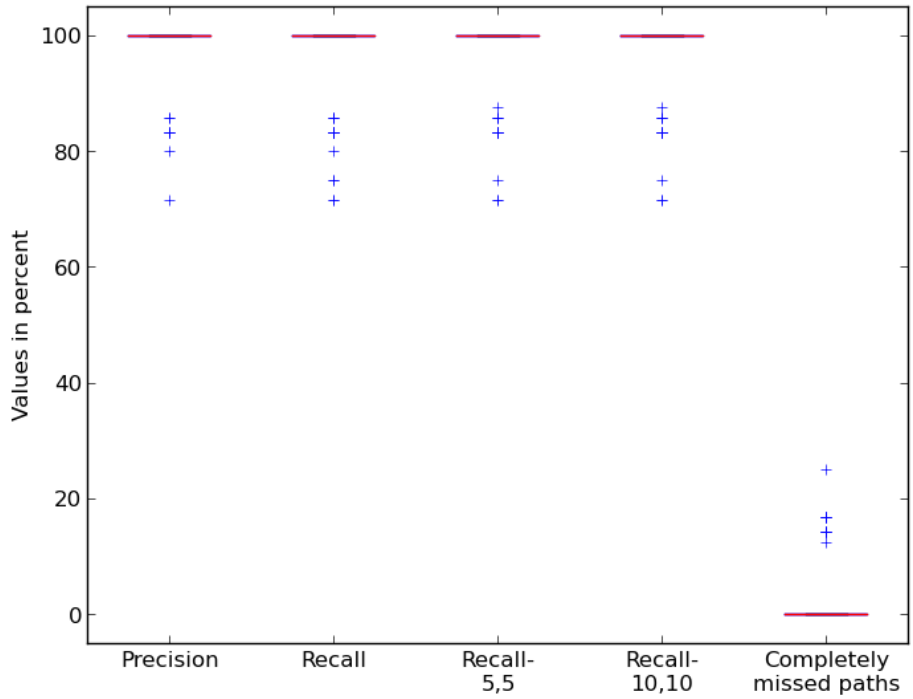
Figure 6.5: Types and Thresholds: Comparison of the basic algorithm (with and without pruning rule) and the IWD heuristic with pruning rule. Durations are given in seconds. The table provides average, 50%-ile, 90%-ile and 99%-ile values.

walking) and around one path of type 3 (much transit, little walking and little car usage). Hence, we consider the set of paths concise and diverse.

However, there is still room for improvement. In the following section we discuss ideas to improve quality and especially reduce query computation.

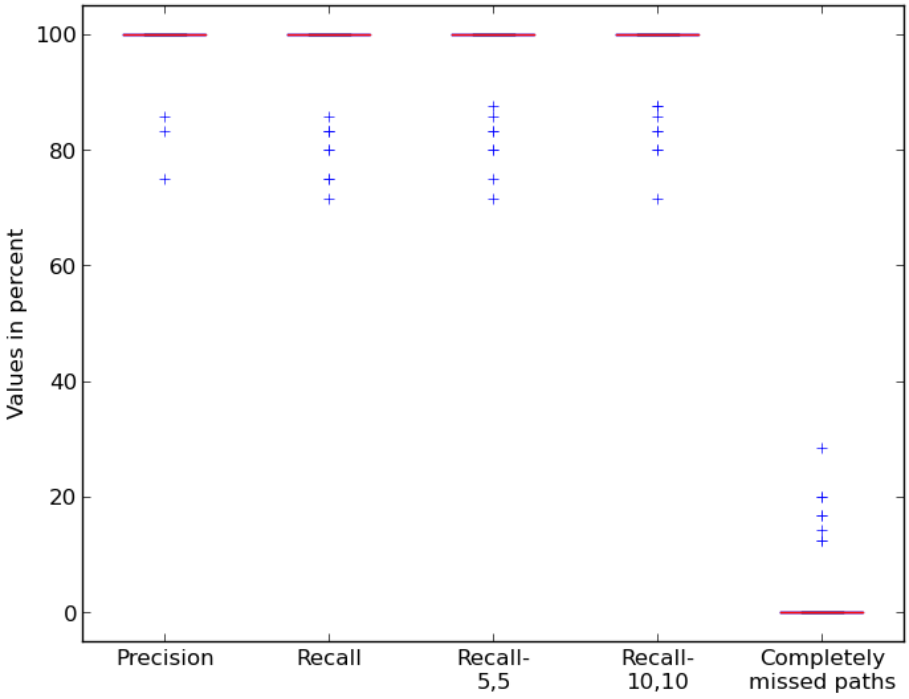


(a) Austin

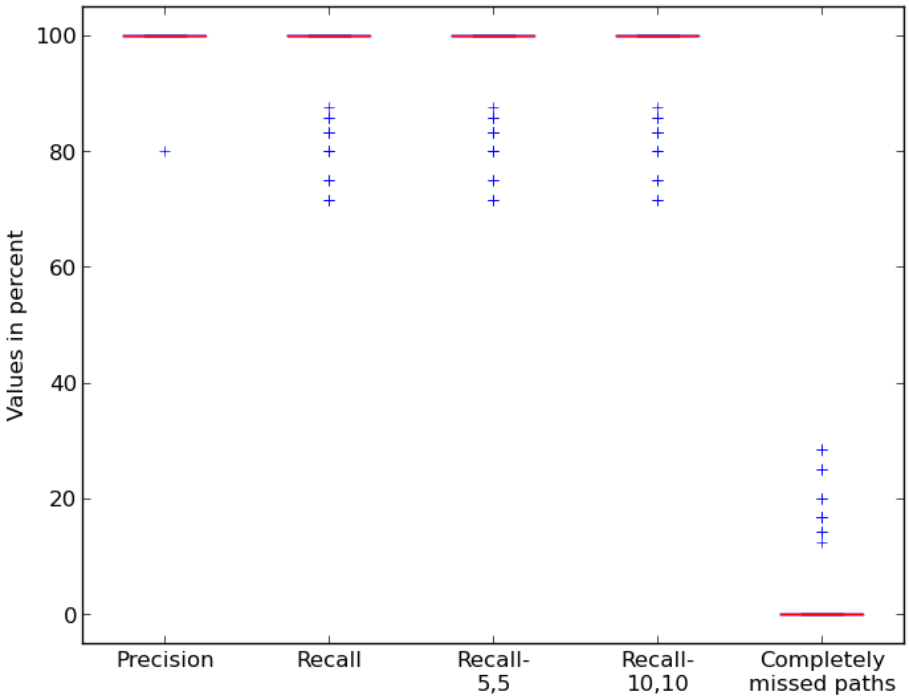


(b) Dallas

Figure 6.6: Types and Thresholds: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets.

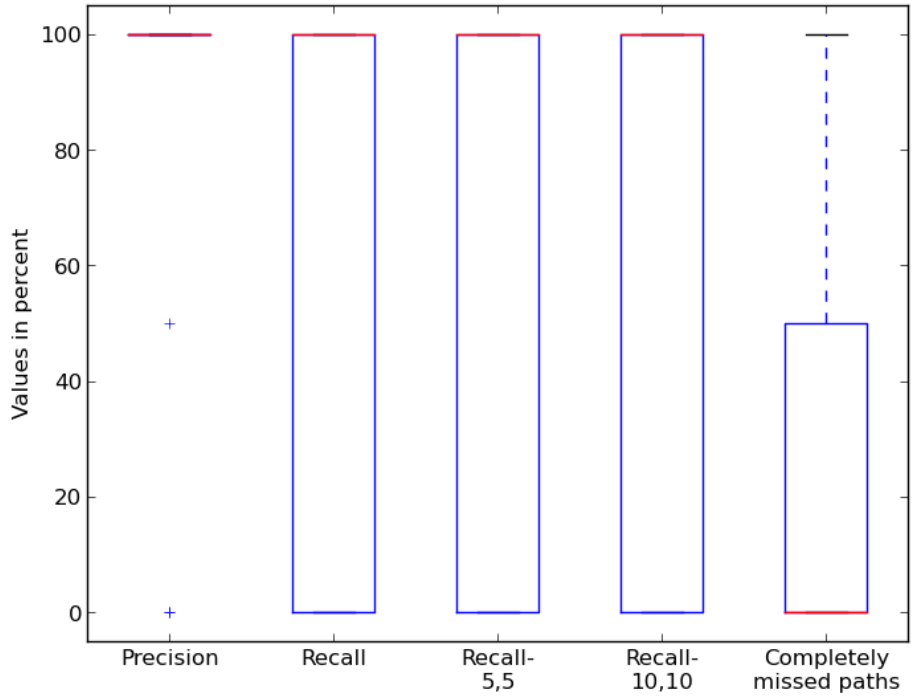


(c) Toronto

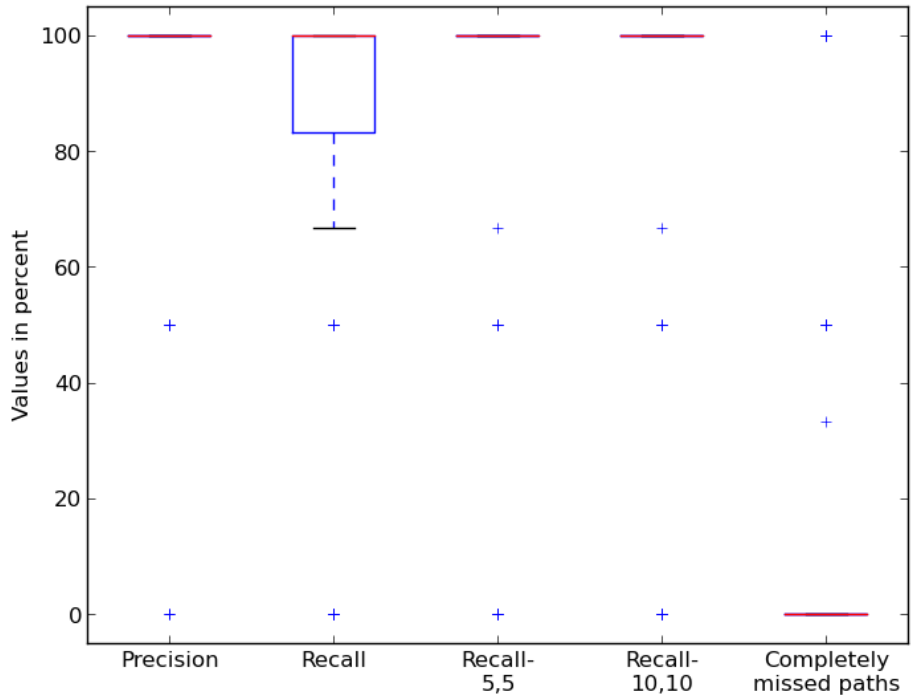


(d) New York

Figure 6.6: Types and Thresholds: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets.

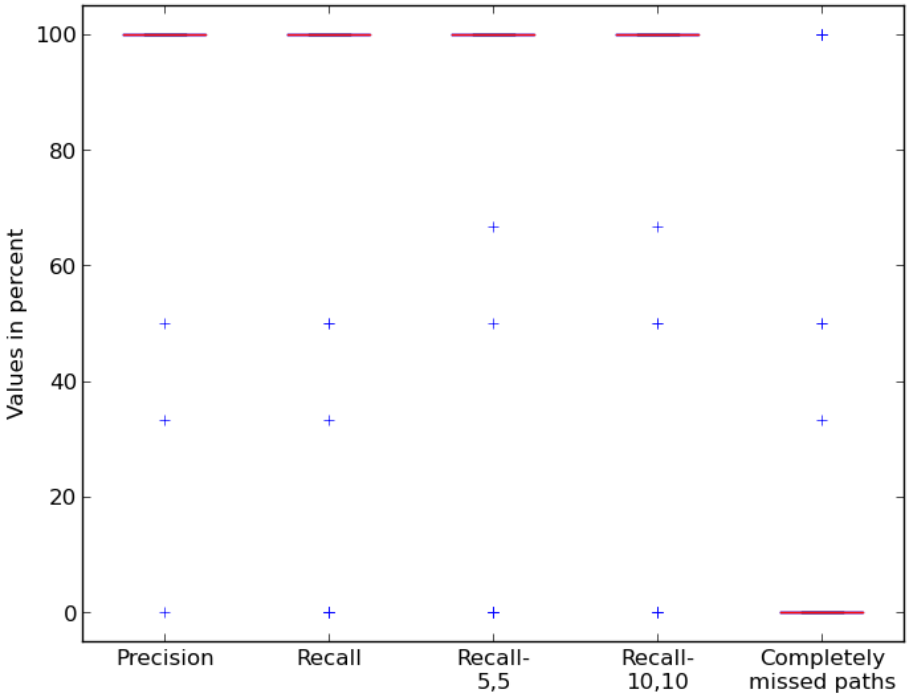


(a) Austin

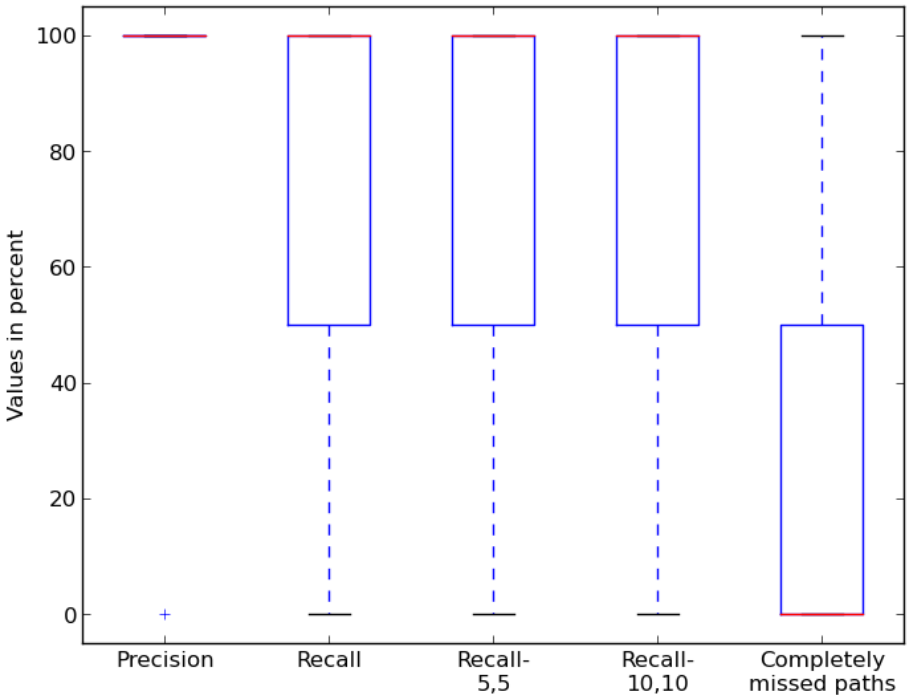


(b) Dallas

Figure 6.7: Types and Thresholds: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets, considering only those paths and queries affected by the IWD heuristic.

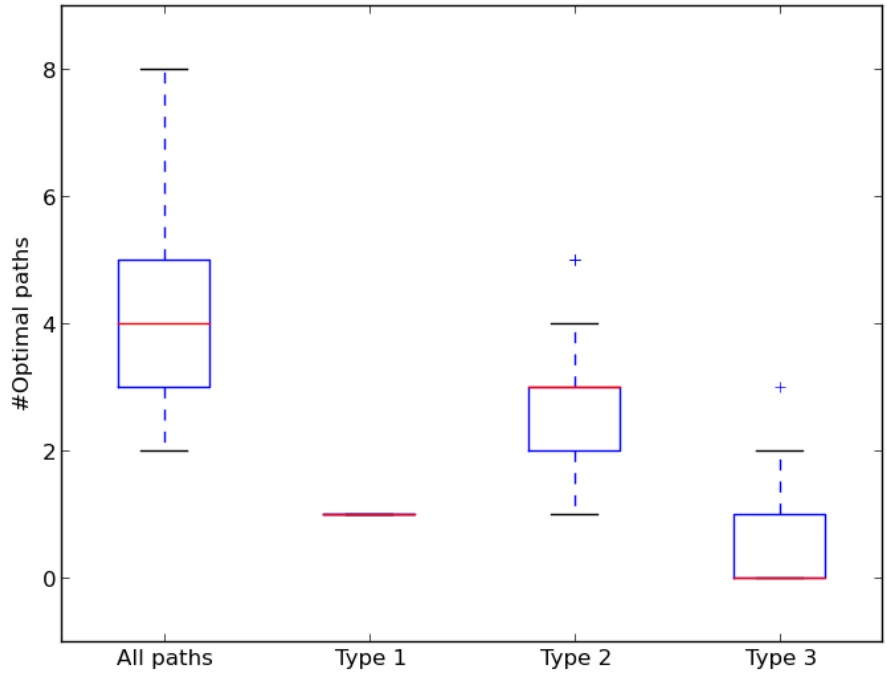


(c) Toronto

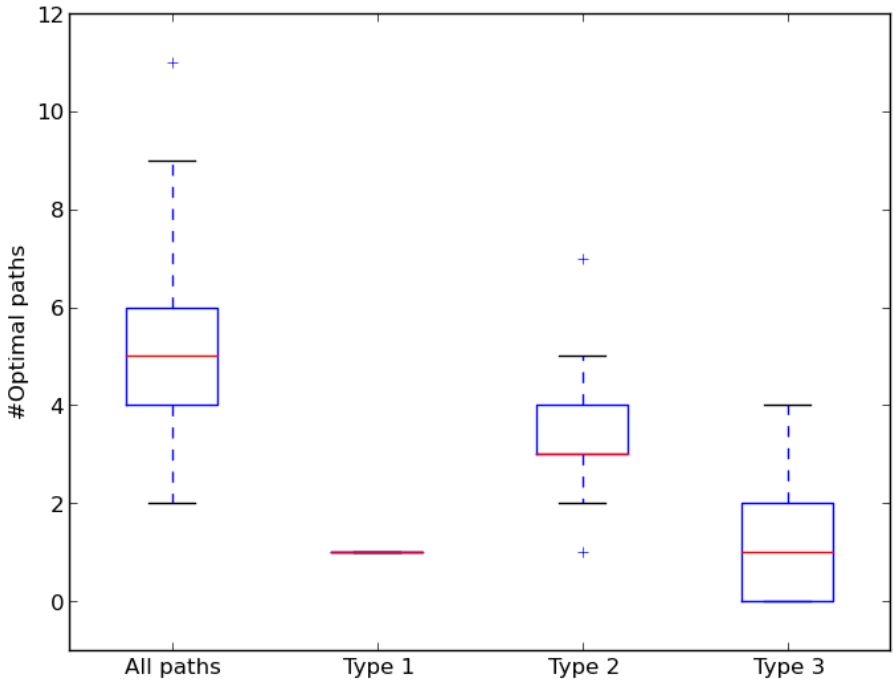


(d) New York

Figure 6.7: Types and Thresholds: Boxplots of Precision, Recall, Recall-5,5, Recall-10,10 and percentage of completely missed paths for all datasets, considering only those paths and queries affected by the IWD heuristic. 39

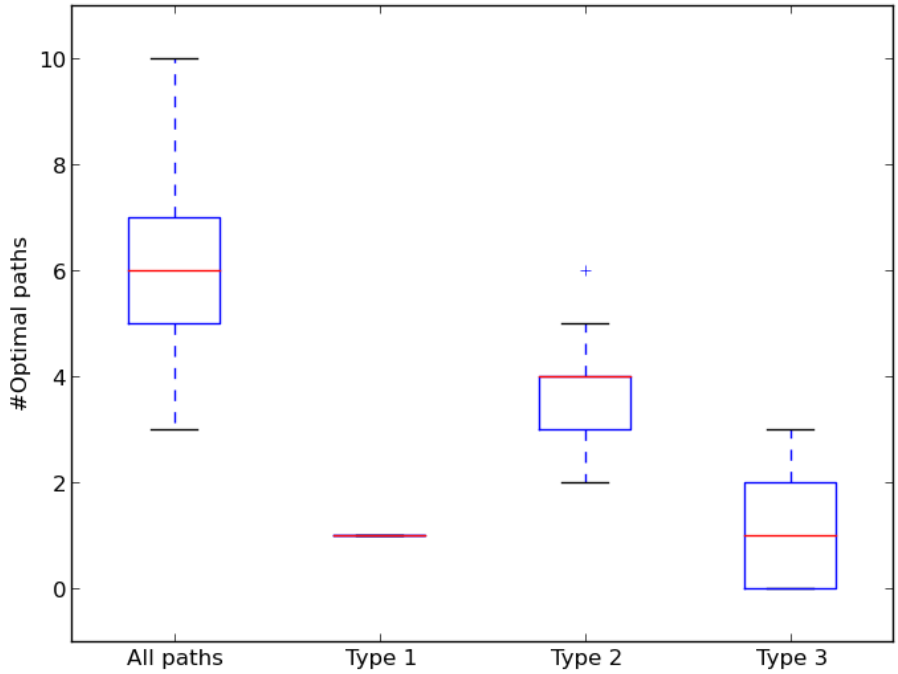


(a) Austin

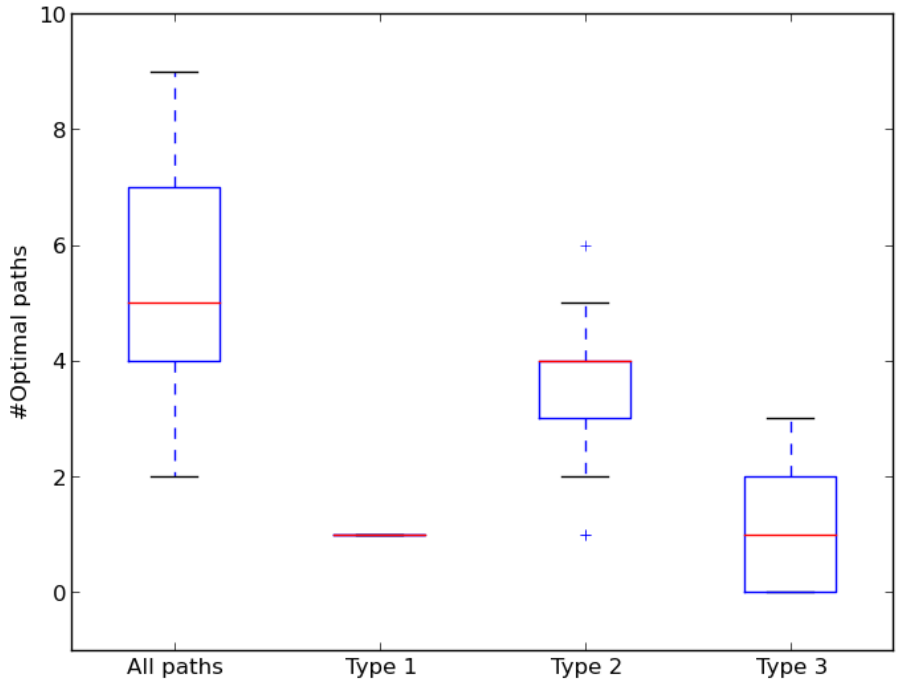


(b) Dallas

Figure 6.8: Types and Thresholds: Boxplots of the number of paths and their distribution to the particular types.



(c) Toronto



(d) New York

Figure 6.8: Types and Thresholds: Boxplots of the number of paths and their distribution to the particular types.

7 Possible Improvements

As experimental results have shown, the approach of using types and thresholds leads to concise and diverse sets of paths. However, certain improvements are still possible. We first shed light on a potential weakness of the approach and discuss a potential way to solve it; afterwards we introduce several ideas of how to improve the query duration to make the algorithm applicable in practice. Moreover, we briefly discuss further issues which were neglected in our approach.

7.1 Quality

Using types and thresholds, the number of returned paths is reasonably low, but there can still be paths in the solution set which, taking into account the presence of the other solution, can be considered unreasonable. The issue is based on using transfer penalty as a Pareto criterion. Consider the following two optimal triples:

duration	**transfer penalty**	**car duration**
1:09:00	4	x
1:10:00	3	y

with either (a) $x = 0 = y$ or (b) $x \neq 0 \neq y$. The latter means that the two triples belong to the same type (in case of (a) type 2 and in case of (b) type 3). Because of that and due to the very similar duration, they can be considered similar with respect to duration and car duration (which has to be either zero or little, for both triples). Therefore, the triple with transfer penalty 4 can be considered superfluous. In section 4.3.1 we introduced a means to solve a similar issue for the Pareto criterion car duration. That is, to solve this problem we could discretise duration in a post-processing step.

7.2 Efficiency

There are several possibilities to improve query durations. First, recall that we use special termination conditions for Dijkstra. The optimality-preserving pruning rules include using pure walking duration as an upper bound for all labels. If the bound is very high, e.g., above 10 hours (which is common for longer-distance queries),

pruning of labels is very limited. The reason in in the time-expanded graph used to represent the transit connections: every node corresponds to a specific time. Thus, nodes with an arbitrary late time might also receive labels, whose extensions can never be parts of optimal paths to the target. These labels are computed unnecessarily. Due to our algorithm (see section 3.3.2) it is not obvious when to stop Dijkstra, since the optimal labels at all core nodes are needed.

Intuitively, great potential lies in terminating Dijkstra as early as possible. This claim is supported by the large amount (roughly 50-75 percent, considering only the core nodes of the road networks) the transit graph constitutes of the whole graph. In the following we describe a way to calculate tighter bounds to prune labels. It is based on the assumption that there exists a final node, where all optimal labels of the target stop are collected. Afterwards we describe an approach how to efficiently fulfil this assumption.

Tighter bounds to prune labels. Assuming there exists a final node where all optimal labels of the target are collected *during* the query, the settled labels of this node can be used as upper bounds for *all* other labels. The reason is that all labels based on such labels will eventually be dominated at the final node.

Extending the graph model to a final node for the target. To apply the tighter bounds described above, we need to modify our current graph model. Recall that currently two Dijkstras (besides the use of Contraction Hierarchies for pure car and walking duration) are executed: Firstly, one from the source, using only non-downward arcs and secondly, one from the target being restricted to the road network graphs limited to non-downward arcs. The optimal costs from the target to all core road network nodes can be expected to be calculated very fast (in the order of milliseconds). Assume these are calculated. Then for each reachable core road network node an arc to the target, with the previously calculated cost, could temporarily be added. We call these arcs *query arcs*, since they are created depending on the specific query. Afterwards, calculating the optimal labels from the source to the target would be possible by executing one Dijkstra from the source in the non-downwards graph, additionally considering the query arcs. Thus, the required final node would exist, where all optimal labels of the target are collected during the query.

While pruning of labels is likely to reduce query times, it is questionable if they will be low enough to allow interactive queries. To achieve the latter, extending Transfer Patterns to our multi-modal scenario seems to be a reasonable idea.

Extending Transfer Patterns to our multi-modal scenario. Recall that Braun [13] introduced an approach for generalizing Transfer Patterns to a multi-modal scenario similar to ours. Besides other questions, generalizing this approach to arbitrary location-to-location queries is only obvious in theory. In practice, computing

full Transfer Patterns for all pairs of locations is infeasible. A solution for traditional Transfer Patterns (i.e., considering only transit and small amounts of walking) is to use hubs. However, for our multi-modal scenario, due to the support of car as a transportation mode, this approach does not seem to be extendable in an obvious way. Further research needs to be done.

Using different graphs and algorithms. Our graph model is not necessarily the ultimate solution. Possibly, time-dependent or even different approaches, like the one proposed in [9], could lead to more efficient computations.

A divide and conquer approach. We already applied a divide and conquer approach in our shortest path algorithm. Computing the set of optimal paths is achieved by calculating the solution of type 1 independently of those of type 2 and 3. We could proceed the dividing further. The solutions of type 2 look very similar to the ones calculable by traditional Transfer Patterns. The only difference is, that in our approach, for type 2 solutions, walking is not limited. Assuming there exists a simple way to extend traditional Transfer Patterns to unlimited walking, we can infer that these solutions could be calculated very fast during query computation. For the remaining solutions of type 3, we could use a modified graph such that the implicit walking duration heuristic would (intuitively; a proof remains to be done) become optimal. The modified graph should look as follows: First recall the properties of type 3 solutions: much transit, little walking and little car; note that we defined little(walking) as a fixed threshold (10 minutes) for each subpath of pure walking. Therefore, in our graph model, we could replace the arcs corresponding to the walking network by arcs between of all pairs of (walking node, station) with walking distance of at most 10 minutes. This could efficiently be done by performing one Dijkstra (using only duration as criterion) from each station to all nodes in the walking network and terminating each Dijkstra early by using the 10 minute threshold. Since all walking connections are now direct connections, it seems that the heuristic of using implicit walking duration will produce equal results like using explicit walking duration. That is, the heuristic would not be a heuristic anymore, but produce optimal results. On this graph, the solutions for type 3 could be computed expecting the amount of labels is considerably reduced, i.e., allowing fast computation of the optimal solutions. Besides applying the introduced pruning rule, the solutions for type 1 and 2, which are both expected to be available in the order of milliseconds, could be used for additional pruning. Further research could shed light on the practicability of this approach.

7.3 Further Extensions

Besides extending the labels to even more criteria, further steps towards meeting every-day reality can be performed. To alleviate the *blue light* effect in the car

network, statistical values (like rush-hours) and real-time traffic news could be taken into account. Moreover, reliability and robustness are important issues to consider.

8 Conclusion

We studied multi-modal route planning, with focus on (almost) unrestricted walking, car and transit networks. We proposed a simple model to combine these networks. Based on the premise that finding a useful cost function would be simple, we began to explore the usefulness of the heuristic of what we call flattening the transit graph. While the heuristic turned out unfavourable, it became clear that finding a useful cost-function for our multi-modal scenario was a challenge in itself, firstly with respect to quality, secondly in terms of efficiency. We rejected the idea to linearly combine different criteria into a single criterion, and focused on Pareto Sets of multiple criteria. The first challenge was to find a set of criteria, leading to more diversified results. We exemplified that the criteria duration, transfer penalty and car duration fulfil this requirement. However, the number of optimal paths would grow towards a number not practical for the use of humans. Moreover, this brought along increased computation time. Since high quality of the computed paths is essential for practical use, we focused on means to improve it, followed by investigations on how to reduce the computation time. With the results in mind presented in [13] we rejected the idea of mainly restricting the graph model to achieve good computation times and especially quality. We primarily focused on means to obtain results of good quality, focusing on ways of post-processing the solution sets towards this goal, with the above mentioned Pareto criteria. As a first approach we introduced discretisation, which decreased the number of results and preserved the diversity of the solution set. While the quality of the results improved, it revealed that some types of solutions were not favourable. To filter out the undesired solutions, we additionally used walking duration as a Pareto criterion and introduced the notion of types. We defined the types based on availability, price and velocity of the different means of transportation. The filtered results are concise and representative. Since using walking duration as an additional Pareto criterion increased the computation times even more, we subsequently focused on means to reduce it. We introduced optimality-preserving ideas and heuristics.

Afterwards we evaluated quality and computation time of the presented algorithms and speed-up techniques on different data sets. Besides showing flattening the transit graph is futile, discretisation proved not sufficient to achieve concise solution sets. The results indicate that using types leads to feasible quality. Moreover, it becomes clear that using the speed-up techniques is necessary. For larger datasets, solely using optimality-preserving methods leads to query times in the order of few minutes. Additionally using heuristics reduces them roughly by one order of magnitude while loosing a small fraction of optimal results. Since this is not sufficient

for interactive queries, we proposed various ideas to further reduce the computation time. Finally, we briefly discussed possible extensions for a more realistic modelling of multi-modal route planning.

Danksagung

Mein Dank geht an Hannah Bast für zahlreiche Diskussionen und Denkanstöße. Darüberhinaus danke ich Josef Brodesser, André Doser und Sabine Storandt dafür, dass sie das Korrekturlesen übernommen haben. Christian Schindelhauer danke ich dafür, dass er die Rolle des Zweitprüfers übernommen hat.

Bibliography

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

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

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

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

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

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

[7] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics, 12, 2007.

[8] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In ESA (1), pages 290–301, 2010.

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

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

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

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

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

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

[15] Ben Strasser. Delay-robust stochastic routing in timetable networks. Master's thesis, Karlsruhe Institute of Technology, August 2012.

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

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

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

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