Dijkstra's Algorithm On–Line: An Empirical Case Study from Public Railroad Transport[1]

Frank Schulz
and
Dorothea Wagner
and
Karsten Weihe
University of Konstanz, Department of Computer & Information Science

Traffic information systems are among the most prominent real–world applications of Dijkstra's algorithm for shortest paths. We consider the scenario of a central information server in the realm of public railroad transport on wide–area networks. Such a system has to process a large number of on–line queries for optimal travel connections in real time. In practice, this problem is usually solved by heuristic variations of Dijkstra's algorithm, which do not guarantee an optimal result. We report results from a pilot study, in which we focused on the travel time as the only optimization criterion. In this study, various speed–up techniques for Dijkstra's algorithm were analysed empirically. This analysis was based on the timetable data of all German trains and on a "snapshot" of half a million customer queries.[2]

1. INTRODUCTION

1.1 Problem.

From a theoretical viewpoint, the problem of finding a shortest path from one node to another one in a graph with edge lengths is satisfactorily solved. In fact, the Fibonacci–heap implementation of Dijkstra's algorithm requires $\mathcal{O}(m + n \log n)$ time, where n is the number of nodes and m the number of edges [6]. However, various practical application scenarios impose restrictions that make this sort of algorithm impractical. For instance, many scenarios impose a strict limitation on space consumption.[3]

[1]©ACM, 2000. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Journal of Experimental Algorithmics, Volume 5(12), 2000, http://www.jea.acm.org

[2]With special courtesy of the *TLC Transport-, Informatik- und Logistik–Consulting GmbH/EVA–Fahrplanzentrum*, a subsidiary of the *Deutsche Bahn AG*.

[3]To give a concrete example: if a traffic information system is to be distributed on CD–Rom or to be run on an embedded system, a naive implementation of Dijkstra's algorithm would typically

Address: University of Konstanz, Box D188, 78457 Konstanz, Germany
{schulz,wagner}@fmi.uni–konstanz.de, weihek@acm.org
http://www.fmi.uni-konstanz.de/~{schulz,wagner,weihe}

In this paper, we consider a different scenario: space consumption is not an issue, but the system has to answer a potentially infinite number of customer queries for optimal travel connections on–line. The real–time restrictions are soft, which basically means that the average response time is more important than the maximum response time. The concrete scenario we have in mind is a central server for public railroad transport, which has to process a large number of queries (*e.g.* a server that is directly accessible by customers through terminals in the train stations or through a web interface).

ASSUMPTION 1.1. *Our computational study is based on two further simplifying assumptions:*

(1) The time tables are identical every day.

(2) The length *of a travel connection is exclusively defined as the travel time (to be specified in greater detail in Definition 1.2 below).*

To achieve Assumption 1.1(1), we simply unite the time tables of all days. Due to this assumption, a customer query need not specify a date.

DEFINITION 1.2. *A* customer query *(or simply* query *for short) is a triple consisting of two stations, A and B, and an* earliest departure time *t. Such a query asks for a connection from A to B that does not start before t and arrives B as early as possible.*

Algorithmic problems of this kind are usually approached heuristically in practice, which means that an optimal result is not guaranteed.

DEFINITION 1.3. *An algorithm for a shortest–path problem is called* distance–preserving *if it produces an optimal connection for every input instance.*

Distance–preserving algorithms are not in wide use in traffic information systems, because the average response time of such an algorithm was perceived to be unacceptable. In a new long–term project, we plan to investigate the question to what extent distance–preserving variants of Dijkstra's algorithm have become competitive on contemporary computer technology. Here we give an experience report from a pilot study, in which we focused on the most fundamental kind of query, namely the one specified in Definition 1.2 (subject to Assumption 1.1(1)).

This scenario is an example of a general problem in the design of practical algorithms, which we discussed in [22]: computational studies based on artificial (*e.g.* random) data do not make much sense, because the characteristics of the real–world data are crucial for the success or failure of an algorithmic approach in a concrete use scenario. Hence, experiments on real–world data are the method of choice.

1.2 Related work.

Various textbooks address speed–up techniques for Dijkstra's algorithm but have no concrete applications in mind, notably [2] and [14]. In [21], Chapter 4, a brief, introductory survey of selected techniques is given (with a strong bias towards the use scenario discussed here). Most work from the scientific side addresses the single–source variant, where a spanning tree of shortest paths from a designated

exceed the available space.

root to all other nodes is to be found. Moreover, the main aspect addressed in work like [5] is the choice of the data structure for the priority queue. In Section 3 below (paragraph on the "search horizon"), we will see that the scenario considered in this paper requires algorithmic approaches that are fundamentally different, and Section 4 will show that the choice of the priority queue is indeed a marginal aspect in view of the goal of our project.

On the other hand, most application–oriented work in this field is commercial, not scientific, and there is only a small number of publications. In fact, we are not aware of any publication especially about algorithms for wide–area railroad traffic information systems.

Some scientific work has been done on *local* public transport. For example, [17] gives some insights into the state of the art. However, local public transport is quite different from wide–area public transport, because the timetables are very regular, and the most powerful speed–up techniques are based on the strict periodicity of the trains, buses, ferries, etc. In contrast, our experience is that the timetables of the national European train companies are not regular enough to gain a significant profit from these techniques.[4]

On the other hand, *private* transport has been extensively investigated in view of wide–area networks. Roughly speaking, this means "routing planners" for cars on city and country maps [9; 1; 4; 20; 12; 18]. This problem is different to ours in that it is two–dimensional, whereas train timetables induce time as a third dimension: due to the lack of periodicity, the earliest departure time is significant in our scenario. In contrast, temporal aspects do not play any role in the work quoted above.[5] So it is not surprising that the research on routing planners has focused on purely geometric techniques.

Some of the application–oriented work quoted above is based on *hierarchical* approaches. This is also one of our strategies (see Section 3.2, strategy "selections of stations"). For example, in [19] this approach is viewed from a database perspective without any specific scenario in mind (though with a bias towards road maps).

For completeness, we mention the work on variants of Dijkstra's algorithm that are intended to efficiently cope with large data in secondary memory. Chapter 9 of [3] gives an introduction to theoretical and practical aspects. As mentioned above ("problem" section in the Introduction), the problems caused by the slow access to secondary memory are beyond the scope of our paper.

Since our main contribution is a computational study, the work on the methodology of this sort of study (e.g. [7; 8; 11; 10; 13; 15; 16]) is also related. We will come back to methodological issues in Section 2.

1.3 Contribution of the paper.

We implemented and evaluated various distance–preserving speed–up techniques for Dijkstra's algorithm. The study is based on all train data (winter period 1996/97) of the Deutsche Bahn AG, the national railroad and train company of Germany.

[4] This experience is supported by personal communications with people from the industry.

[5] In principle, temporal aspects would also be relevant for the private transport, for example, the distinction between "rush hours" and other times of the day. However, this sort of temporal aspect is quite different from ours and, to our knowledge, not scientifically invesitigated either.

The processed queries are a "snapshot" of the central *Hafas*[6] server of the Deutsche Bahn AG, in which all queries of customers of all ticket offices in Germany were recorded over several hours.

In particular, an input instance consists of a fixed and a variable part:

—*Fixed*: all input instances include the same set of time tables.

—*Variable*: the query, that is, departure station, arrival station, and earliest departure time, varies from input instance to input instance.

The result of this snapshot comprises more than half a million queries, which might suffice for a representative analysis of this particular service (assuming that the typical query profile of customers does not vary dramatically from day to day).

Due to the above–mentioned insight that the periodicity of the timetables is not a promising base for algorithmic approaches, the question is particularly interesting whether geometric techniques like those in routing planners are successful, although the scenario has geometric *and* temporal characteristics. We will see that this question can indeed be answered in the affirmative.

1.4 Overview.

In Section 2 we will discuss the general rationale of our experimental set–up. Then, in Section 3, we will briefly summarize the algorithmic techniques that we evaluated. Finally, in Section 4, we will present and discuss the results of the computational study.

2. EMPIRICAL METHODOLOGY

As mentioned in the Introduction, the question to be answered by our computational study is very specific:

> Are distance–preserving variants of Dijkstra's algorithm (in the sense of Definition 1.3) competitive on contemporary computer technology in the use scenario addressed in this paper?

The notion "competitive" is here used in a very informal sense, which is probably *the* sense of interest from an applied viewpoint. Roughly speaking, we regard an algorithm as competitive if it would probably not become a bottleneck operation in a typical central server. Such a fuzzy notion does not allow far–reaching conclusions. We believe that our conclusions (which we will give in detail in Section 4 and in summary in Section 5) are modest enough to be safely based on our usage of competitiveness.

The question addressed in our paper is quite different from the questions typically addressed in methodological work such as [7; 8; 11; 10; 13; 15; 16]. In fact, to our knowledge, the general focus in the literature is on systematic performance comparisons of different algorithms without or with only loose relation to concrete applications. The overwhelming majority of computational studies in theoretical computer science primarily addresses this kind of question. The method of choice here is the evaluation of the individual algorithms on artificial (mainly random)

[6]Hafas is a trademark of the Hacon Ingenieurgesellschaft mbH, Hannover, Germany.

input data, which is a good base for the systematic variation of parameters, and a statistical analysis of the computational results.

In contrast, the question we address in this paper is tightly bound to a concrete application scenario. The profile of the input data is very specific, but a sufficient specification of this profile in terms of parameters such as the number of nodes/edges or the average/maximum node degree does not seem to be in our reach.[7] Thus, there is no base for the creation of artificial, yet realistic data. On the other hand, a snapshot of half a million real customer queries, which is the base of our study, is certainly sufficient for a profound answer to the above question. Hence, we will restrict our attention to real–world data.

Note that the question addressed in this paper is binary (yes/no). As mentioned in the Introduction, the real–time constraints are rather soft. This means that the overall average response time, taken over the total set of all queries, is the most important piece of information to decide upon yes or no. However, beyond this very specific question we also regard an estimation of the scope of the result as important. For instance, such a large body of response times, if well analysed, should allow a reliable prediction how the average response time changes if the average geographic distance of departure and arrival station changes. To give a concrete example: the percentage of long–distance queries is significantly higher for the web service than for the ticket office service, from which our snapshot was taken.

To estimate the effect of variations like this, we will also consider the interdependence between response times and two parameters, the Euclidean distance between departure and arrival station and the minimal total travel time computed in response to the respective query. In Section 4 we will see that this dependency is surprisingly strong, which means that we may assume a high level of confidence for predictions.

As mentioned in the Introduction, we implemented and evaluated various speed–up techniques for Dijkstra's algorithm. One notable exception is the *bidirected–search technique*. This technique does not really apply in our scenario, because the target node of the search depends on the arrival time and is thus not known in advance (see Section 3, paragraph on "bidirected search," for details). However, one can imagine a variation of the bidirected–search technique, which does not rely on the exact arrival time but on an upper bound. To answer the question whether this class of modified bidirected–search techniques is potentially competitive at all, we implemented and evaluated an idealized variant, which indeed does rely on the exact arrival time and thus yields a lower bound on the potential performance of this class of speed–up techniques.

[7] Nonetheless, a comparison of the algorithms from Section 3 based on random data and a conventional statistical analysis as advocated by some of the above–mentioned papers may be of interest in its own right. However, such an analysis does not contribute to our specific question and is thus beyond the scope of this paper.

3. ALGORITHMS

3.1 Fundamental Algorithmic Approach

3.1.1 *Train graph..* From Assumption 1.1 recall our restriction to time tables that do not vary from day to day. Due to this restriction, we only need a train network of 24 hours.

The arrival or departure of a train at a station will be called an *event*. The train graph contains one node for every event. Two events v and w are connected by a directed edge (v, w) if v represents the departure of a train at some station and w represents the very next arrival of this train at some other station. On the other hand, two successive events at the same station are connected by an edge (in positive time direction), which means that every station is represented in the graph by a cycle through all of its events (the cycle is closed by a turn–around edge at midnight).

In each case, the length of an edge is the time difference (in minutes) of the events represented by its endnodes. Obviously, a query then amounts to finding a shortest path from the earliest event at the departure station not before the earliest departure time to an arbitrary arrival event at the arrival station.

The data contains $933,066$ events on $6,961$ stations. Consequently, there are $933,066 \cdot 3/2 = 1,399,599$ edges in the graph.

3.1.2 *Priority queue..* Dijkstra's algorithm relies on a priority queue, which manages the events on the current "frontier line" of the search. As mentioned above, the best general worst–case bound, $\mathcal{O}(m + n \log n)$, is obtained from Fibonacci heaps, where n is again the number of events and m the number of edges. We do not use a Fibonacci heap but a normal heap (also often called a 2–*heap*), which yields an $\mathcal{O}((n + m) \log n)$ bound [6]. Since $m \in \mathcal{O}(n)$ in train graphs, both bounds reduce to $\mathcal{O}(n \log n)$.

As an alternative to heaps, we also implemented the *Dial variant* as described in [2]. Basically, this means that the priority queue is realized by an array of buckets with a cyclically moving array index. The events of the frontier line are distributed among the buckets, and it is guaranteed that the very next non–empty bucket after the current array index always contains the candidates to be processed next. The asymptotic worst–case bound of the Dial implementation of Dijkstra's algorithm is $\mathcal{O}(m + nL)$, where L is the largest length of an edge.

3.1.3 *Search horizon..* Of course, we deviate from the "textbook version" of Dijkstra's algorithm in that we do not compute the distance of every event from the source event but terminate the algorithm immediately once the first (and thus optimal) event at the arrival station is processed. The most fundamental distance–preserving speed–up technique for our scenario is then a reduction of the search to a (hopefully) small part of the graph, which contains all relevant events. Figs. 1–4 demonstrate that such a reduction is crucial. In fact, they reveal that for the majority of all queries only small fractions of the total area and time horizon are relevant.

To our knowledge, some commercial implementations remove events and edges from the graph (more or less heuristically, *i.e.* losing optimality of distances) before the search itself takes place. In contrast, we aim at an evaluation of distance–

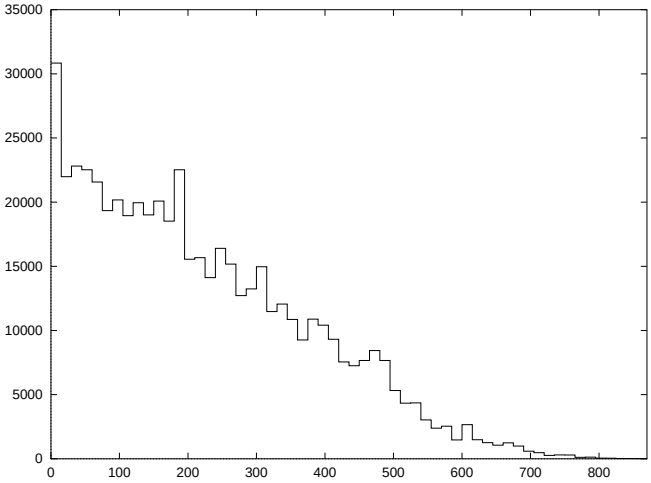

Fig. 1. The frequency distribution histogram of the queries from the snapshot according to the Euclidean distance between the departure station and the arrival station (granularity: 15 kilometers).

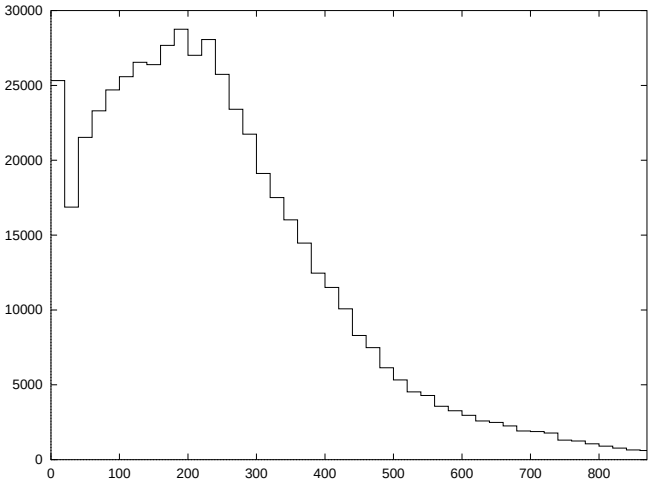

Fig. 2. Like Fig. 1 except that the abscissa now denotes the minimal travel time in minutes (granularity: 20 minutes).

preserving strategies, so our approach is quite different. First of all, we apply the amortization technique discussed in [21] to obtain a sublinear expected run time per query.[8] The only obstacle to sublinearity is the initialization of all events with infinite distance labels in the beginning of the textbook algorithm; in fact, Figs. 1–4

[8]The term "sublinear expected run time" is here used in a rather informal, experience–oriented way and not to be understood in a formal probabilistic sense.

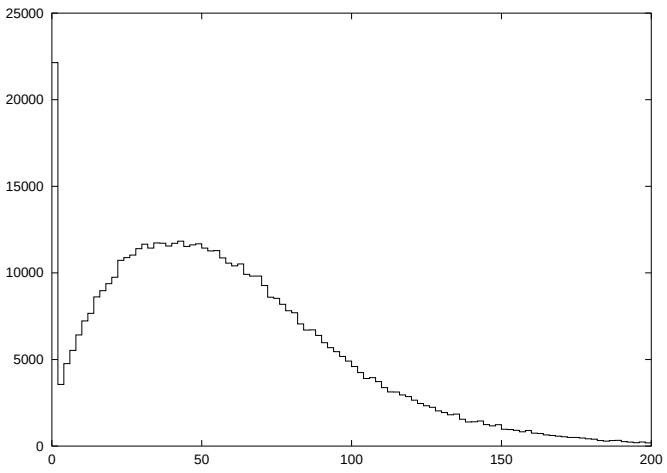

Fig. 3. Like Fig. 1 except that the abscissa now denotes the number of edges in the shortest path (granularity: 10 edges).

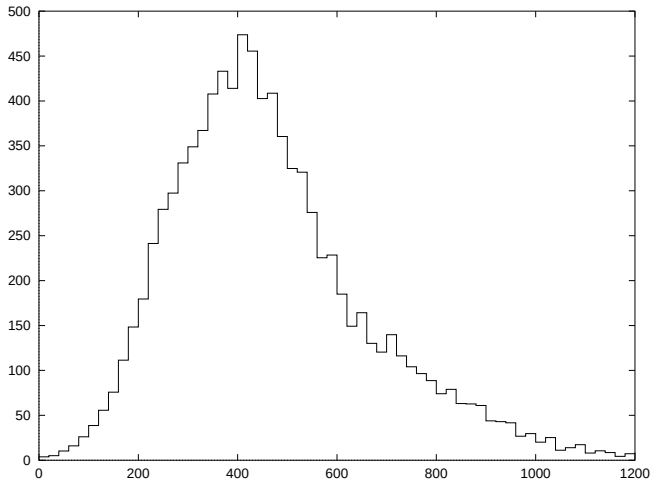

Fig. 4. For each minimal travel time (granularity: 20 minutes) the total CPU times of all queries yielding this value of the travel time are summed up to reveal which range of travel times contributes most of the total CPU time required for all queries.

strongly suggest that on average the main loop of the algorithm only processes a very small fraction of the graph until the arrival station is seen and the algorithm terminates.

As described in [21], every event is given an additional *time stamp*, which stores the number of the query in which it was reached in the main loop. Whenever an event is reached, its time stamp is updated accordingly. This update properly increases the time stamp if and only if this is the first time that the node is seen

in the current search. This is exactly the situation in which the current distance label should be infinity. Hence, if the time stamp is increased, the distance label is regarded as infinite, otherwise the value of the distance label is taken as is. Consequently, there is no need for an expensive initialization phase, and no event outside the "search horizon" of the main loop is hit at all.

It might be worth mentioning an equivalent strategy to avoid the initialization step: An additional data structure, e.g. a linked list, is used to store the nodes with non-infinite distance labels. Then, all non-infinite distance labels can be reset to infinity after each query.

3.2 Algorithmic Variations

The good performance of the following additional techniques relies on the general design decisions introduced in Section 3.1. All of these techniques are independent of each other in the sense that an arbitrary selection of them may be applied simultaneously.

3.2.1 *Angle restriction..* This technique additionally relies on the coordinates associated with the individual stations. In a preprocessing step, we apply Dijkstra's algorithm to each event to compute shortest paths from this event to all other stations.[9] The results are not stored (this would require too much space) but only used to compute two values $\alpha(v,w)$ and $\beta(v,w)$ for each edge (v,w) that corresponds to a train hop. These $2 \cdot n/2$ values (n again the number of events) are stored and then used in the on–line system.

More specifically, these values are to be interpreted as angles in the plane. See Fig. 5. Let (v,w) be an edge from some event v to some other event w, and let s_v and s_w be the stations to which events v and w belong, respectively. Then the values $\alpha(v,w)$ and $\beta(v,w)$ stored for edge (v,w) span a circle sector with center s_v in the plane, where, say, $\alpha(v,w)$ is the right leg. The values $\alpha(v,w)$ and $\beta(v,w)$ are determined such that this circle sector has the following meaning: if the shortest path from event v to some event u at some station s' contains (v,w), then s' has to be in this circle sector. Subject to this condition, the difference of the angles, $(\beta(v,w)-\alpha(v,w)) \bmod 2\pi$, is minimum. Clearly, the brute–force application of Dijkstra's algorithm in the preprocessing allows one to compute this specific, narrowest possible circle sector for each edge.

Consequently, edge (v,w) may be ignored by the search if the arrival station is *not* in the circle sector of this edge. The restriction of the search to edges whose circle sectors contain the arrival station is the strategy that we will call "angle restriction" in the following.

PROPOSITION 3.1. *Due to the specific construction of the values $\alpha(\cdot,\cdot)$ and $\beta(\cdot,\cdot)$, the strategy "angle restriction" is distance–preserving in the sense of Definition 1.3.*

3.2.2 *Selection of stations..* The basic idea behind this technique is similarly implemented in various routing planners for the private transport [9; 1; 4; 12].

[9] This preprocessing takes several hours, which is absolutely acceptable in the practical setting from which this research originated, namely generating time tables once for every winter/summer period. Hence, there is no need to optimize the preprocessing.

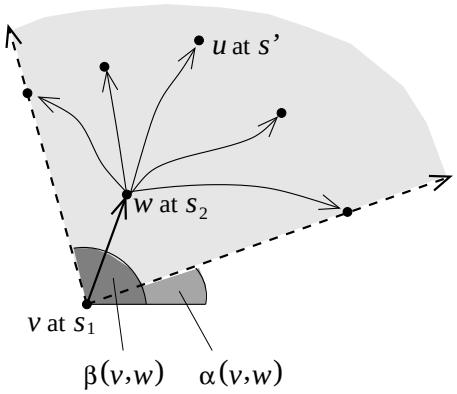

Fig. 5. The circle sector spanned by $\alpha(v, w)$ and $\beta(v, w)$ contains all stations s' such that there is an event u at s' whose shortest path from v starts with (v, w).

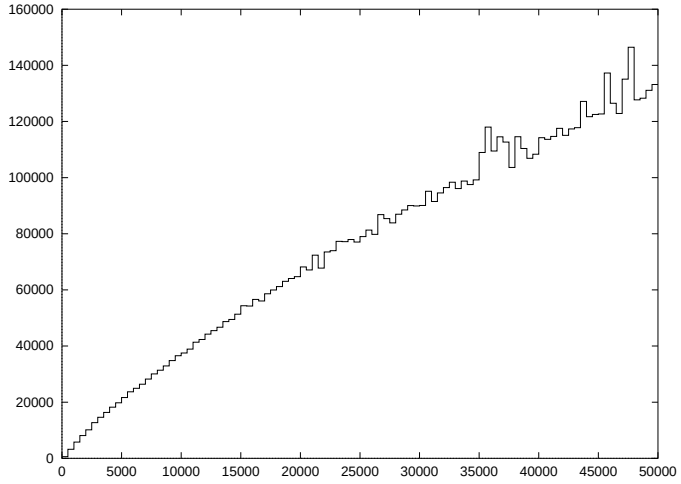

Fig. 6. The relation of the number of edges hit without (abscissa) and with (ordinate) strategy angle selection (granularity: 500 edges).

In general, a certain small set of nodes (*i.e.* events) is selected, which means events in our setting. However, we will only select sets of nodes such that either every node from a station s is selected, or none of the nodes from station s. More specifically, in the data available to us, every station is assigned an "importance number," which is intended to rank its degree of "centrality" in the railroad network. For our computational study, we select the 225 stations in the highest categories (see Fig. 10). These stations induce $95,423$ events in total, which constitute our selection of events.

Let $G = (V, E)$ denote the (directed) train graph. In particular, V is the set of all events. Moreover, let $\tilde{V}$ be the set of selected nodes. The strategy is based on what we will call *replaceable paths* in the following. Let v, $w \in V$ be two events, which may or may not belong to $\tilde{V}$. A path from v to w in G will be called *replaceable*, if

no node of the path (except for the endnodes, v and w, themselves) belongs to $\tilde{V}$.

Based on this notion, we construct an auxiliary directed graph $G' = (V, E')$ on V. For v, $w \in V$, there is a directed edge $(v, w) \in E'$ if, and only if, there is a replaceable path from v to w in G. The length $\ell(v, w)$ of an edge $(v, w) \in E'$ in G' is then defined as the minimum length of a replaceable path from v to w in G.

The auxiliary graph G' and these edge weights are also constructed once and for all in a preprocessing step (which only takes a few minutes). Each query is then answered by the computation of a shortest path in the auxiliary graph G', and this path corresponds to a shortest path in the train graph G.

We implement this general approach as follows. First of all, note that it is not necessary to reconstruct the path in G that corresponds to the shortest path computed in G'. In fact, what we really want to have from the computation is a sequence of trains and the stations where to change train. This data can be attached to the edges of the auxiliary graph in an even more compact (less redundant) and thus more efficiently evaluable fashion than to the edges of the train graph.

As mentioned above, we select a set of stations, and the events of these stations are the selected events. Clearly, there is a trade–off. Roughly speaking, the smaller the number of selected stations is, the larger the resulting connected components are and, even worse, the larger the number of selected stations neighboring to a component. Since the number of edges depends on the latter number in a quadratic fashion, an improvement of performance due to a rigorous reduction of stations is soon outweighed by the tremendous increase in the number of edges.

It has turned out that in our setting, a minor refinement of this strategy is sufficient to overcome this trade–off. For this, let u, v, and w be three selected events such that edges (u, v), (v, w), and (u, w) exist in the auxiliary graph. If $\ell(u, v)+\ell(v, w) \leq \ell(u, w)$, then edge (u, w) is dropped in the auxiliary graph. Again, the optimal distance is preserved. The number of edges grows only moderately after this modification, so quite a small set of selected stations becomes feasible.

The selected stations induce $95,423$ events in total, which means that the number of events is approximately reduced by a factor of 10, and the number of stations is reduced by a factor of 31. This discrepancy between these two factors is not surprising, because central stations are typically met by more trains than other stations.

PROPOSITION 3.2. *The strategy "selection of stations" is distance–preserving in the sense of Definition 1.3.*

3.2.3 *Goal–directed search..* This strategy is based on one real number $b(s)$ for each station s, which must be a lower bound on the minimal travel time from s to the arrival station. Let $(v, w) \in E$ be an edge, and let s_v and s_w be the stations of v and w, respectively. The values $b(s_v)$ and $b(s_w)$ must additionally satisfy $b(s_v) - b(s_w) \leq \ell(v, w)$. The length $\ell(v, w)$ of (v, w) is then replaced by the modified length $\ell'(v, w) := \ell(v, w) - b(s_v) + b(s_w) \geq 0$.

It can be shown that the shortest paths with respect to the length function $\ell'(\cdot, \cdot)$ are identical with the shortest paths with respect to the original length function $\ell(\cdot, \cdot)$. Hence, we may alternately compute a shortest path with respect to $\ell'(\cdot, \cdot)$. The hope is that the "frontier" of the search horizon approaches the arrival station faster when the modified edge lengths are used. See [14] for a more systematic

discussion.

Roughly speaking, the tighter the lower bounds $s(\cdot)$ are, the larger the speed–up effect will be. The base of our lower bounds is the maximum speed of all trains as a transformation factor. The lower bound for an arbitrary station with respect to a given target station is the Euclidean distance of these two stations divided by this factor. These values are computable "on the fly" during the search.

PROPOSITION 3.3. *The strategy "goal–directed search" is distance–preserving in the sense of Definition 1.3.*

3.2.4 *Bidirected search..* This strategy "merges" two applications of Dijkstra's algorithm into one loop. More specifically, one search starts from the source event and goes forward as usual, whereas the other search starts from the target event and traverses every edge in backward direction. The algorithm is finished once an event was labeled permanently by both searches. A shortest path from the source event to the target event is then easily constructed from the results of the two searches up to this moment. We refer to [2; 14] for details.

This strategy is not directly applicable in our scenario, because the target event is not known until the search is finished. Nonetheless, one can imagine variations of this strategy, in which the backward search indeed starts from the arrival station, however, not from the optimal arrival time but from an upper bound on this value.

To evaluate the general potential of this strategy in view of our application scenario, we implemented an idealized variant. More specifically, we indeed use the exact optimal arrival time as a start point for the backward search. Clearly, this gives a lower bound on the performance of all of these variations of bidirected search.[10] Since we only consider an idealized strategy here, it does not make sense to evaluate the CPU times achieved by this strategy. Hence, we will restrict the evaluation of this strategy to operation counts (see Section 4).

3.2.5 *Combination of strategies..* In principle, all of these strategies can be freely combined. In that, the strategy "selection of stations" can be combined with the strategy "angle restrictions" in two ways, namely the angle restrictions can be computed for the auxiliary graph, or they can be computed for the train graph and simply taken over for the auxiliary graph.

4. ANALYSIS OF THE ALGORITHMIC PERFORMANCE

The experiments were performed on a SUN Sparc Enterprise 5000, and the code was written in C++ using the GNU compiler.

Tables 1 and 2 present a summarizing comparison of all combinations of strategies. Since every node in the train graph has a degree of three, the total number of algorithmic steps is asymptotically dominated by the number of operations inside the priority queue. In other words, these operations are representative operation counts in the sense of [2], Sect. 18. More specifically, for a heap the number of exchange operations is representative, whereas for the Dial variant the number of cyclic increments of the moving array index plus the number of the bucket-operations "insert" and "remove" is representative. The average total number per

[10] Except for pathological cases that are imagineable but might hardly occur in reality.

query of these operations are listed in the last part of Tables 1 and 2. As mentioned in Section 2, we regard these average numbers as the most important piece of information in view of our central yes/no question.

For both implementations of the priority queue, the CPU times imply the same strong ranking of combinations of strategies. Figs. 9–12 might give a visual impression why this ranking is so unambiguous. Moreover, the discrepancy between the heap and the Dial implementation in terms of CPU times also decreases roughly from row to row.

Our experience with several versions of the code is that the exact CPU times are strongly sensitive to the details of the implementation, but the general tendency was maintained and seems to be reliable. In particular, the main question raised in this paper (whether distance–preserving techniques in the sense of Definition 1.3 are competitive) can be safely answered in the affirmative at least for the restriction of the problem to the total travel time as the only optimization criterion.

However, a detailed look at the results is more insightful (and necessary to conclude predictions for scenarios with different profiles as discussed in Section 2). Fig. 6 shows that there is a very strong correlation between the number of edges hit with/without strategy "angle restriction." On the other hand, Fig. 7 shows a detailed analysis of one particular, exemplary combination of strategies. A comparison of the diagrams in Fig. 7 reveals an interesting effect, which is also found in the analogous diagrams of the other combinations: the CPU times of both the heap and the Dial implementation "look" linear in the number of nodes hit by the search.[11]

This correlation is so strong and the variance is so small that corresponding diagrams in Fig. 7 look almost identical. Figure 8 reveals the cause. In other words, in both cases the operations on the priority queue come close to constant time on average, even when the average is taken over each query separately.

Finally, the bidirected–search technique required, on average, 5606 nodes, 9497 edges, an operation count of 7531 for the Dial implementation, and an operation count of 45826 for the heap implementation. As mentioned in the paragraph on bidirected search in Section 3, it does not make sense to take CPU times here, because an idealized algorithm was implemented.

Since one can safely assume that the above results for this idealized algorithm are a lower bound on the real performance of this technique in our scenario, they give some evidence that bidirected search would be a reasonable choice (assuming that tight upper bounds on the exact arrival times can be found efficiently), but might not be the favorite choice.

In Section 2, we raised the question whether the results allow reliable predictions for similar scenarios such as a web server, where a larger fraction of all queries asks for long–distance connections. The small variance encourages us to conclude (with caution) that our results are scalable in an approximately linear fashion to

[11] From the work on this study we gained the experience that the difference between $\Theta(n)$ and $\Theta(n \log n)$ may be very small and hidden by the "noise" of the data. Hence, the claim that one can observe a linear behavior does not really contradict the claim that the asymptotic complexity is $\Theta(n \log n)$. In other words, the above claim that the run time be linear is to be taken with a pinch of salt.

Selection of stations	Angle restriction	CPU heap	CPU Dial
no	no	0.310	0.103
no	yes	0.036	0.018
yes	no	0.027	0.012
yes	yes (train)	0.007	0.005
yes	yes (auxiliary)	0.005	0.003

Selection of stations	Angle restriction	Nodes	Edges
no	no	17576	31820
no	yes	4744	10026
yes	no	2114	3684
yes	yes (train)	1140	2737
yes	yes (auxiliary)	993	2378

Selection of stations	Angle restriction	Ops. heap	Ops. Dial
no	no	246255	23866
no	yes	24526	3334
yes	no	26304	3660
yes	yes (train)	4973	1197
yes	yes (auxiliary)	3191	932

Table 1. A summary of all computational results for the individual combinations of the techniques "selection of stations" and "angle restrictions." The entries "train" and "auxiliary" in column #2 refer to the graph in which the angles were computed (see the last paragraph in Section 3). The columns #3–#4 give the average over all queries of the snapshot. More specifically, the first table gives the average raw CPU times, the second table the average number of nodes and edges hit by the search, and the third table the average operation counts.

scenarios like that.

As announced in the Introduction, the choice of the data structure for the priority queue seems to be a detail of minor relevance. The difference between the heap and the Dial implementation looks significant at first glance, however, this seems to be one of the aspects which are not stable but highly dependent on implementation details.

5. CONCLUSION AND OUTLOOK

The outcome of this study suggests that geometric speed–up techniques are a good basis for the computation of provably optimal connections in railroad traffic information systems. The question raised in this paper is answered for the total travel time: the best combinations of strategies are by far faster than is currently needed in practice. The success of geometry–based strategies is a bit surprising, because the underlying data is not purely geometric in nature.

Another surprising outcome of the study is that both the normal heap and the Dial data structure only require an (amortized) constant time per operation, whereas the worst–case bound is logarithmic for heaps and even linear for Dials. Note that no amortization over a set of queries must be applied to obtain a constant time per operation; the variance is small enough that the average run time per operation within a single query can essentially be regarded as bounded by a

Selection of stations	Angle restriction	CPU heap	CPU Dial
no	no	0.204	0.072
no	yes	0.022	0.013
yes	no	0.019	0.010
yes	yes (train)	0.006	0.004
yes	yes (auxiliary)	0.005	0.003

Selection of stations	Angle restriction	Nodes	Edges
no	no	11072	20046
no	yes	3415	7120
yes	no	1437	2465
yes	yes (train)	856	2002
yes	yes (auxiliary)	754	1765

Selection of stations	Angle restriction	Ops. heap	Ops. Dial
no	no	151274	15274
no	yes	16990	2430
yes	no	18123	2595
yes	yes (train)	3502	961
yes	yes (auxiliary)	2282	751

Table 2. Like Table 1 except that now the strategy "goal–directed search" is additionally applied in all cases.

constant. Due to the fact that the variance is negligible, a conventional statistical analysis would not make any sense.

The choice of algorithms (and of minor details of their implementation) is justified a posteriori in view of our main question. Implementing and comparing further speed–up strategies is certainly an interesting topic in its own right. However, the goal of our research effort is different: to go far beyond the restricted scenario of Assumption 1.1 and to see which algorithms will stand this ultimate test.

The minimal travel time is certainly an empirical research topic in its own right, not only because it is the most fundamental objective in practice. However, a practical algorithm must consider further criteria and restrictions. For example, the ticket costs and the number of train changes are also important objectives. Moreover, certain trains do not operate every day (contrasting Assumption 1.1(1)), and certain kinds of tickets are not valid for all trains, so it should be possible to exclude train connections in a query. A satisfactory compromise must be found between the speed of the algorithm and the quality of the result. Thus, the problem is not purely technical anymore but also involves "business rules," which are usually *very* informal.

In the future, an extensive requirements analysis will be necessary, which means that the work will be no longer purely "algorithmical" in nature. Such an analysis must be very detailed because otherwise there is no hope to match the *real* problem. Unfortunately, there is high evidence that the general problems addressed in [22] will become virulent here: a sufficiently simple formal model that captures all relevant details does not seem to be in our reach, and many details are "volatile"

in the sense that they may change time and again in unforeseen ways. Future research will show whether these conflicting criteria can be simultaneously fulfilled satisfactorily.

REFERENCES

[1] R. Agrawal and H. Jagadish. Algorithms for searching massive graphs. *IEEE Transact. Knowledge and Data Eng.*, 6:225–238, 1994.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice–Hall, 1993.

[3] S. Albers, A. Crauser, and K. Mehlhorn. *Algorithmen für sehr große Datenmengen*. MPI Saarbrücken, 1997. [12]

[4] A. Car and A. Frank. Modelling a hierarchy of space applied to large road networks. In *Proc. Int. Worksh. Adv. Research Geogr. Inform. Syst. (IGIS '94)*, pages 15–24, 1994.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw–Hill, 1994.

[7] J. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42:201–212, 1994.

[8] J. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[9] K. Ishikawa, M. Ogawa, S. Azume, and T. Ito. Map navigation software of the electro multivision of the '91 toyota soarer. In *IEEE Int. Conf. Vehicle Navig. Inform. Syst. (VNIS '91)*, pages 463–473, 1991.

[10] H.F. Jackson, P.T. Boggs, S.G. Nash, and S. Powell. Guidelines for reporting results of computational experiments—report of the ad hoc committee. *Mathematical Programming*, 49:413–425, 1991.

[11] D.S. Johnson. A theoretician's guide to the experimental analysis of algorithms, 1996. `http://www.research.att.com/~dsj/papers/exper.ps`.

[12] S. Jung and S. Pramanik. Hiti graph model of topographical road maps in navigation systems. In *Proc. 12th IEEE Int. Conf. Data Eng.*, pages 76–84, 1996.

[13] C.Y. Lee, J. Bard, M.L. Pinedo, and W. Wilhelm. Guidelines for reporting computational results in iie transactions. *IEE Transactions*, 25:121–123, 1993.

[14] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, 1990.

[15] C. McGeoch. Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. *ACM Computing Surveys*, 24(2):195–212, 1992.

[16] C. McGeoch. Toward an experimental method of algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996.

[17] T. Preuss and J.-H. Syrbe. An integrated traffic information system. In *Proc. 6th Int. Conf. Appl. Computer Networking in Architecture, Construction, Design, Civil Eng., and Urban Planning (europIA '97)*, 1997.

[18] J. Shapiro, J. Waxman, and D. Nir. Level graphs and approximate shortest path algorithms. *Network*, 22:691–717, 1992.

[19] S. Shekhar, A. Fetterer, and G. Goyal. Materialization trade–offs in hierarchical shortest path algorithms. In *Proc. Int. Symp. Large Spatial Databases, Springer Lecture Notes in Computer Science 1262*, pages 94–111, 1997.

[20] S. Shekhar, A. Kohli, and M. Coyle. Path computation algorithms for advanced traveler information system (atis). In *Proc. 9th IEEE Int. Conf. Data Eng.*, pages 31–39, 1993.

[21] K. Weihe. Reuse of algorithms — still a challenge to object–oriented programming. In *Proc. 12th ACM Symp. Object–Oriented Programming, Systems, Languages, and Applications (OOPSLA '97)*, pages 34–48, 1997.

[12] Available from `http://www.mpi-sb.mpg.de/units/ag1/extcomp.html`. The lecture notes are in German, however, the chapter on shortest paths (Chapter 9) is in English.

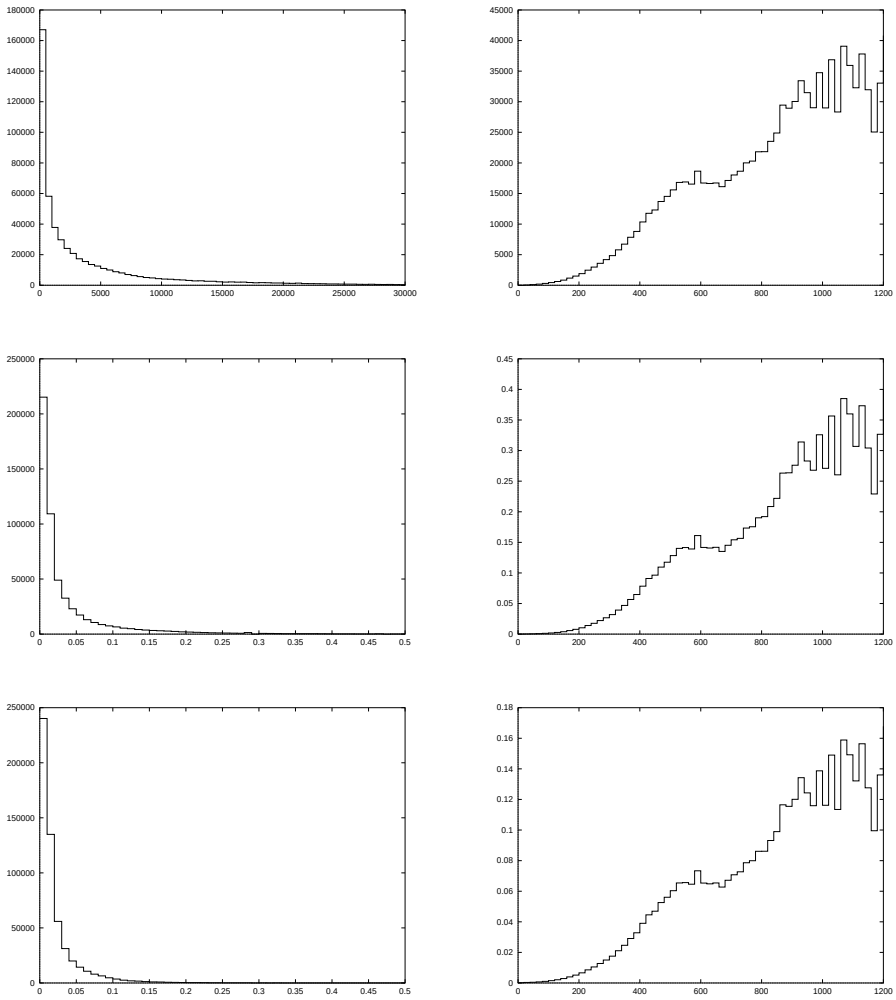

Fig. 7. An exemplary sequence of diagrams for one particular combination of strategies: "angle restriction" is applied, but the other strategies are not. First column: the frequency distribution histogram of all queries in the snapshot according to (a) the number of nodes met by the search (granularity: 500 nodes), (b) the CPU time for the heap implementation and (c) for the Dial implementation (granularity: 10 milliseconds). Second column: the average of (a) the number of nodes met and (b/c) the CPU times for the heap/Dial variant taken over all queries with roughly the same resulting total travel times (granularity: 20 minutes). The strong resemblance of the diagrams in each row nicely demonstrates the linear behavior of both priority–queue implementations.

[22] K. Weihe, U. Brandes, A. Liebers, M. Müller-Hannemann, D. Wagner, and T. Willhalm. Empirical design of geometric algorithms. In *Proc. 15th ACM Symp. Comp. Geometry (SCG '99)*, pages 86–94, 1999.

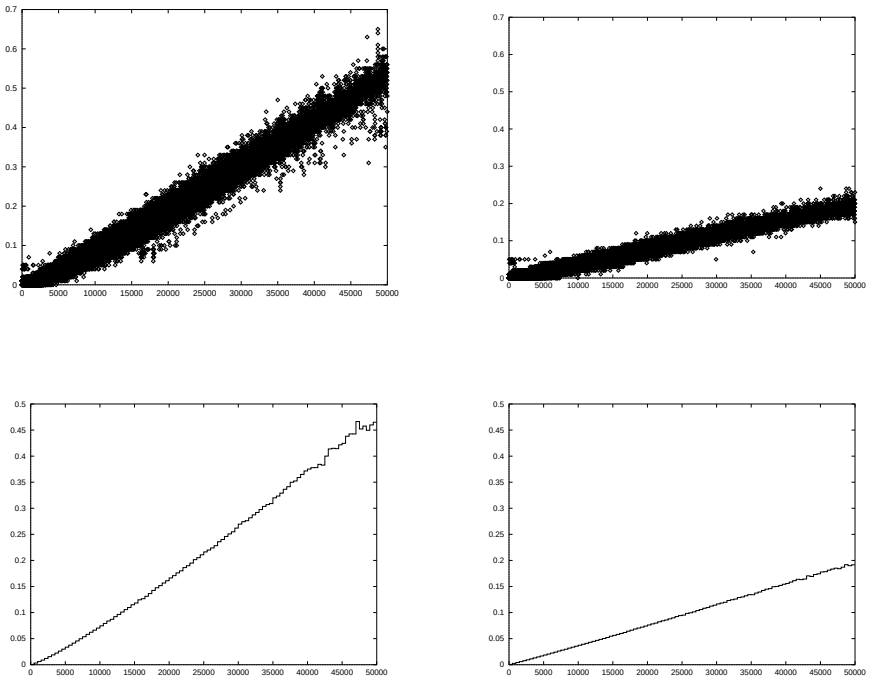

Fig. 8. The relation between the number of nodes hit by the search (abscissa) and the cpu time in seconds for the heap (left) and the Dial (right) implementation. The first row shows one point for every query, the second row condenses this information to histograms (granularity: 500 nodes).

Fig. 9. The left picture shows the edges hit by Dijkstra's algorithm from Berlin Main East Station until the arrival station Frankfurt/Main Main Station is reached. In the right picture, the strategy "angle selection" was applied to the same query.

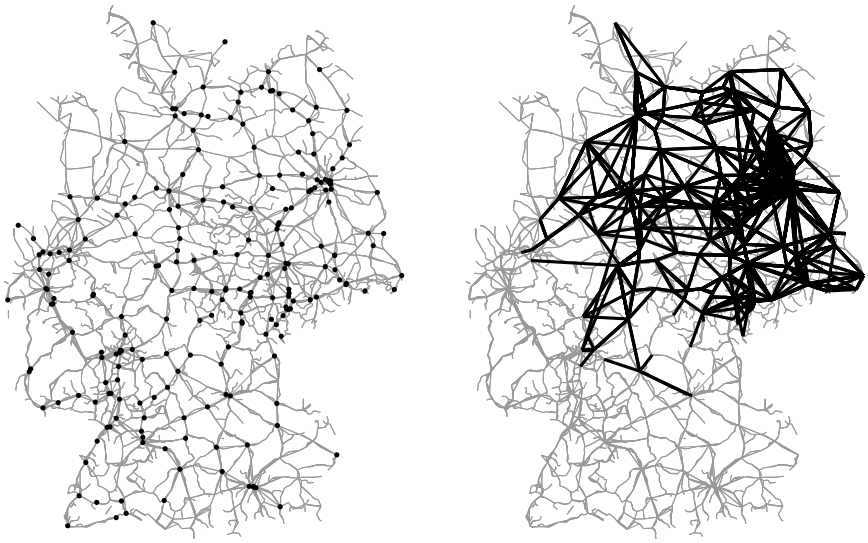

Fig. 10. The left picture shows the 225 stations selected for the study on strategy "selection of stations," and the right picture shows the edges hit by this strategy for the same query as in Figure 9. In the right picture, the original train graph is shown in the background, whereas the highlighted edges are the edges of the *auxiliary* graph hit by the search.

Fig. 11. Both pictures refer to the same query as in Fig. 9. The strategies "selection of stations" and "angle restriction" are applied simultaneously with angles computed from the train graph (left), and with angles computed from the auxiliary graph itself (right). Like in the previous picture, the original train graph is shown in the background, and the highlighted edges are the edges of the auxiliary graph hit by the search.

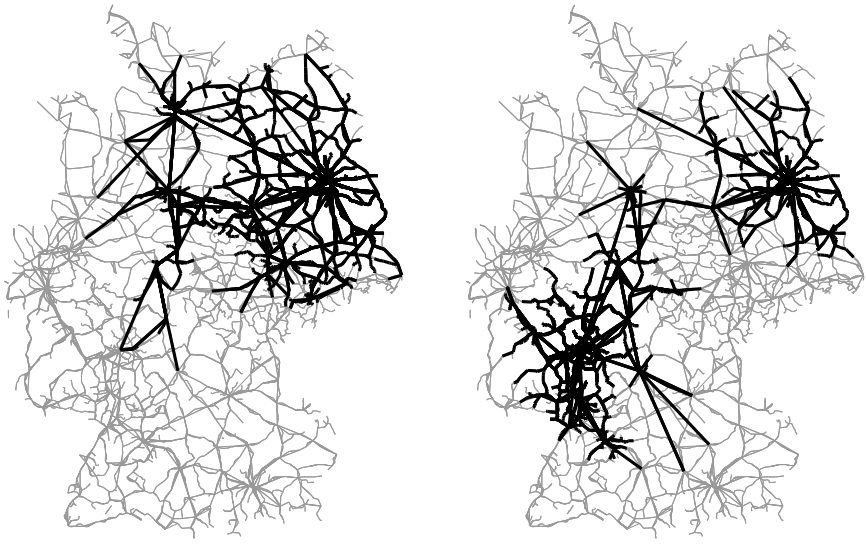

Fig. 12. Both pictures refer to the same query as in Fig. 9. They show the results of the strategy "goal–directed search" and "bidirected search," respectively.