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

Frank Schulz, Dorothea Wagner, and Karsten Weihe

Universität Konstanz, Fakultät für Mathematik und Informatik
Fach D188, 78457 Konstanz, Germany
{schulz,wagner,weihe}@fmi.uni-konstanz.de
http://www.fmi.uni-konstanz.de/~{schulz,wagner,weihe}

Abstract. 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 in real time. In practice, this problem is usually solved by heuristical variations of Dijkstra's algorithm, which do not guarantee optimality. We report results from a pilot study, in which we focused on the travel time as the only optimization criterion. In this study, various optimality–preserving speed–up techniques for Dijkstra's algorithm were analyzed empirically. This analysis was based on the timetable data of all German trains and on a "snapshot" of half a million customer queries.[1]

1 Introduction

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 [8]. However, various practical application scenarios impose restrictions that make this algorithm impractical. For instance, many scenarios impose a strict limitation on space consumption.[2]

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 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

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

[2] 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 exceed the available space.

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 WWW interface).

Algorithmic problems of this kind are usually approached heuristically in practice, because the average response time of optimal algorithms seems to be inacceptable. In a new long–term project, we investigate the question to what extent optimality–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 queries: find the fastest connection from some station A to some station B subject to a given *earliest departure time*.

This scenario is an example of a general problem in the design of practical algorithms, which we discussed in [14]: 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.

Related work. Various textbooks address speed–up techniques for Dijkstra's algorithm but have no concrete applications in mind, notably [4] and [10]. In [13], 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 root to all other nodes is to be found. Moreover, the main aspect addressed in work like [6] is the choice of the data structure for the priority queue. In Section 2 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 3 will show that the choice of the priority queue is a marginal aspect here.

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, [12] 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, busses, 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.[3]

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 [2,3,5,7,9,11]. This problem is different to

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

ours in that it is two–dimensional, whereas train timetables induce the 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.[4] So it is not surprising that the research has focused on purely geometric techniques.

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 [1] gives an introduction to theoretical and practical aspects. As mentioned above, the problems caused by the slow access to secondary memory are beyond the scope of our paper.

Contribution of the paper. We implemented and tested various optimality–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. The processed queries are a "snapshot" of the central *Hafas*[5] server of the Deutsche Bahn AG, in which all queries of customers were recorded over several hours. The result of this snapshot comprises more than half a million queries, which might suffice for a representative analysis (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.

2 Algorithms

Train graph. 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 $v \to w$ 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 of the events represented by its endnodes. Obviously, a query then amounts to finding a shortest path from the earliest event at the start station not before the earliest departure time to an arbitrary arrival event at the destination.

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.

[4] 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.

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

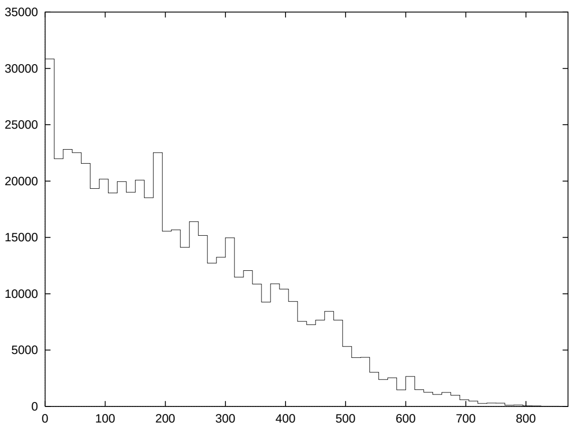

Fig. 1. The frequency distribution histogram of the queries from the "snapshot" according to the Euclidean distance between the start station and the destination (granularity: 15 kilometers).

Priority queue. Dijkstra's algorithm relies on a priority queue, which manages the nodes 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 nodes 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 [8]. 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 [4]. Basically, this means that the priority queue is realized by an array of buckets with a cyclically moving array index. The nodes 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.

Search horizon. Of course, we deviate from the "textbook version" of Dijkstra's algorithm in that we do not compute the distance of every node from the start node but terminate the algorithm immediately once the first (and thus optimal) event at the destination is processed. The most fundamental optimality–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. 1–3 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 nodes and edges from the graph (more or less heuristically, *i.e.* losing optimality) before the search itself takes place. In contrast, we aim at an evaluation of optimality–

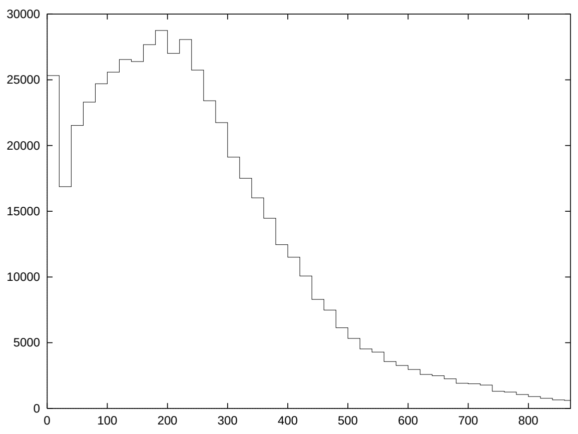

Fig. 2. Like Fig. 1 except that the abscissa now denotes the minimal travel time in minutes (granularity: 20 minutes).

preserving strategies, so our approach is quite different. First of all, we apply the amortization technique discussed in [13] to obtain a sublinear expected run time per query. The only obstacle to sublinearity is the initialization of all nodes with infinite distance labels in the beginning of the textbook algorithm; in fact, Figs. 1 and 2 strongly suggest that on average the main loop of the algorithm only processes a very small fraction of the graph until the destination is seen and the algorithm terminates.

As described in [13], every node is given an additional *time stamp*, which stores the number of the query in which it was reached in the main loop. Whenever a node is reached, its time stamp is updated accordingly. If this update properly increases the time stamp, 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.

The following two additional techniques rely on this general outline. They are independent of each other in the sense that one of them may be applied alone, or both of them may be applied simultaneously.

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.[6] The results are not stored (this would require too much space) but only used to compute two values α and β for each edge. These $2 \cdot m$ values are stored and then used in the on–line system. More specifically, these values are

[6] This preprocessing takes several hours, which is absolutely acceptable in practice. Hence, there is no need to optimize the preprocessing.

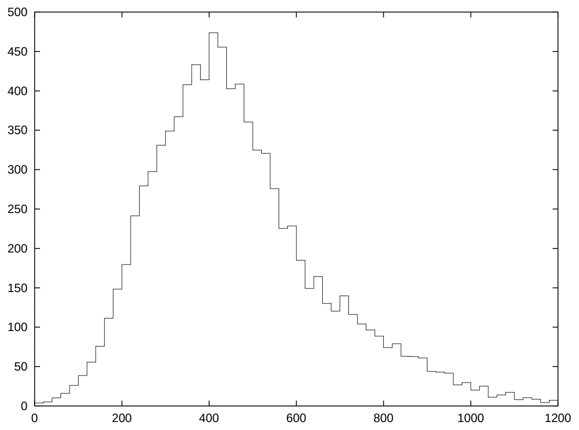

Fig. 3. For each minimal travel time (granularity: 20 minutes) the total CPU times of all queries yielding this value of the travel time are summed up to reveal which range of travel times contributes most of the total CPU time.

to be interpreted as angles in the plane. Let $v \rightarrow w$ be an edge, and let s be the station of event v. Then the values α and β stored for this edge span a circle sector with center s. The meaning is this: if the shortest path from event v to some station s' contains $v \rightarrow w$, then s' is in this circle sector. Clearly, the brute–force application of Dijkstra's algorithm in the preprocessing allows one to compute the narrowest possible circle sector for each edge subject to this constraint.

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

Selection of stations. The basic idea behind this technique is similarly implemented in various routing planners for the private transport [2,3,5,9]. A certain small set of nodes is selected. For two selected nodes v and w, there is an edge $v \rightarrow w$ if, and only if, there is a path from v to w in the train graph such that no internal node of this path belongs to the selected ones. In other words, every connected component of non–selected stations (plus the neighboring selected stations) is replaced by a directed graph defined on the neighboring selected stations. This constitutes an additional, *auxiliary* graph.

The length $\ell(v, w)$ of an edge $v \rightarrow w$ in this auxiliary graph is defined as the minimum length of a path from v to w in the train graph that contains no selected nodes besides v and w. The auxiliary graph 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, and this path corresponds to a shortest path in the train graph.

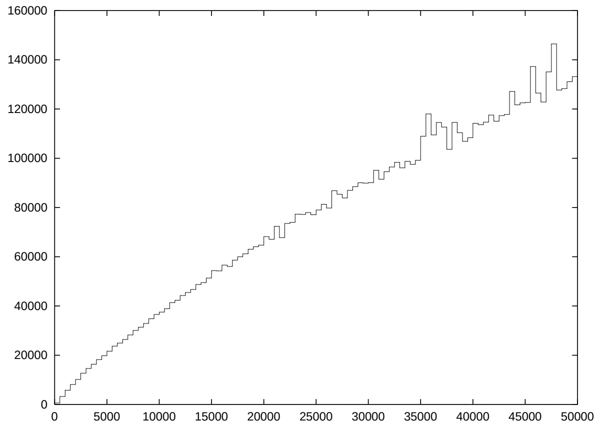

Fig. 4. The relation of the number of edges hit without (abscissa) and with (ordinate) strategy angle selection (granularity: 500 edges).

We implement this general approach as follows. First of all, note that it is not necessary to reconstruct the path in the train graph that corresponds to the shortest path computed in the auxiliary graph. 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.

We select a set of stations, and the events of these stations are the selected nodes. 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 necessary and sufficient to overcome this trade–off. For this, let u, v, and w be three selected events such that edges $u \rightarrow v$, $v \rightarrow w$, and $u \rightarrow w$ exist in the auxiliary graph. If $\ell(u, v) + \ell(v, w) \leq \ell(u, w)$, then edge $u \rightarrow w$ is dropped in the auxiliary graph. Again, optimality is preserved. The number of edges grows only moderately after this modification, so a quite small set of selected stations becomes feasible.

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. The computational study is based on the 225 stations in the highest categories (see Fig. 7). These 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 marginal stations.

Combination of both strategies. In principle, these two strategies can be combined 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. Not surprising, we will see in the next section that the former strategy outperforms the latter one.

3 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.

Table 1 presents a summarizing comparison of all combinations of strategies. Note that 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 [4], 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 is representative. The average total number per query of these operations are listed in the last part of Table 1.

For both implementations of the priority queue, the CPU times imply the same strong ranking of strategies. Figs. 7 and 8 might give a visual impression why this ranking is so unambiguous. Moreover, the discrepancy between the heap and the dial implementation also decreases roughly from row to row. This is not surprising: the overhead of the heap should be positively correlated with the size of the heap, which is significantly reduced by both strategies.

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 is maintained and seems to be reliable. In particular, the main question raised in this paper (whether optimality–preserving techniques 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. Fig. 4 shows that there is a very strong linear correlation between the number of edges hit with/without strategy "angle restriction." On the other hand, Fig. 5 shows a detailed analysis of one particular, exemplary combination of strategies. A comparison of these diagrams 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 are linear in the number of nodes hit by the search. This correlation is so strong and the variance is so small that corresponding diagrams in Fig. 5 look almost identical. Figure 6 reveals the cause.

In other words, in both cases the operations on the priority queue take constant time on average, even when the average is taken over each query separately! This is in great contrast to the asymptotic worst–case complexity of these data structures.

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	train	0.007	0.005
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	train	1140	2737
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	train	4973	1197
yes	auxiliary	3191	932

Table 1. A summary of all computational results for the individual combinations of techniques. The entries "train" and "auxiliary" in column #2 refer to the graph in which the angles were computed (see the last paragraph in Section 2). 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.

4 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. This success 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 constant. Due to the fact that the variance is negligible, a "classical" statistical analysis would not make any sense.

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, 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 [14] 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. S. Albers, A. Crauser, and K. Mehlhorn: *Algorithmen für sehr große Datenmengen.* MPI Saarbrücken (1997).[7]
2. K. Ishikawa, M. Ogawa, S. Azume, and T. Ito: *Map Navigation Software of the Electro Multivision of the '91 Toyota Soarer.* IEEE Int. Conf. Vehicle Navig. Inform. Syst. (VNIS '91), 463–473.
3. R. Agrawal and H. Jagadish: *Algorithms for Searching Massive Graphs.* IEEE Transact. Knowledge and Data Eng. 6 (1994), 225–238.
4. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin: *Network Flows.* Prentice–Hall, 1993.
5. A. Car and A. Frank: *Modelling a Hierarchy of Space Applied to Large Road Networks.* Proc. Int. Worksh. Adv. Research Geogr. Inform. Syst. (IGIS '94), 15–24.
6. B.V. Cherkassky, A.V. Goldberg und T. Radzik: *Shortest Paths Algorithms: Theory and Experimental Evaluation.* Mathematical Programming 73 (1996), 129–174.
7. S. Shekhar, A. Kohli, and M. Coyle: *Path Computation Algorithms for Advanced Traveler Information System (ATIS).* Proc. 9th IEEE Int. Conf. Data Eng. (1993), 31–39.
8. T.H. Cormen, C.E. Leiserson, and R.L. Rivest: *Introduction to Algorithms.* MIT Press and McGraw–Hill, 1994.
9. S. Jung and S. Pramanik: *HiTi Graph Model of Topographical Road Maps in Navigation Systems.* Proc. 12th IEEE Int. Conf. Data Eng. (1996), 76–84.
10. T. Lengauer: *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, 1990.

[7] 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.

11. J. Shapiro, J. Waxman, and D. Nir: *Level Graphs and Approximate Shortest Path Algorithms.* Network 22 (1992), 691–717.
12. T. Preuss and J.-H. Syrbe: *An Integrated Traffic Information System.* Proc. 6th Int. Conf. Appl. Computer Networking in Architecture, Construction, Design, Civil Eng., and Urban Planning (europIA '97).
13. K. Weihe: *Reuse of Algorithms — Still a Challenge to Object–Oriented Programming.* Proc. 12th ACM Symp. Object–Oriented Programming, Systems, Languages, and Applications (OOPSLA '97), 34–48.
14. K. Weihe, U. Brandes, A. Liebers, M. Müller–Hannemann, D. Wagner, and T. Willhalm: *Empirical Design of Geometric Algorithms.* To appear in the Proc. 15th ACM Symp. Comp. Geometry (SCG '99).

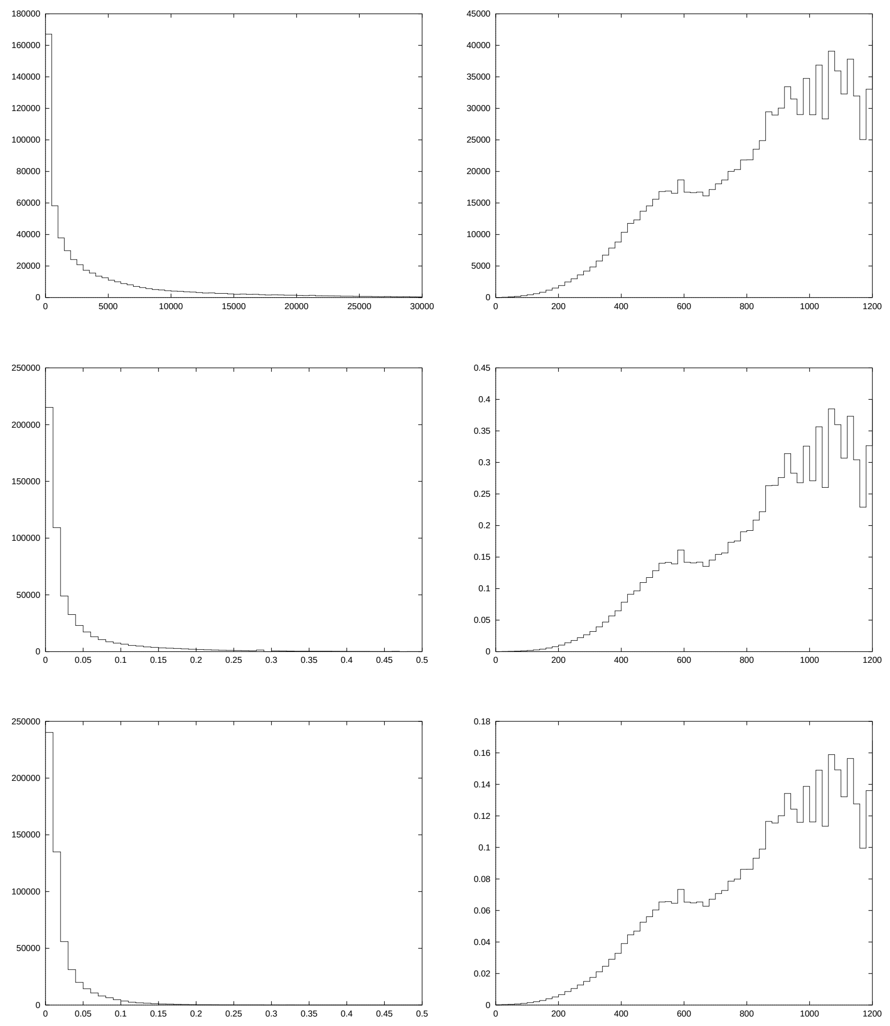

Fig. 5. An exemplary sequence of diagrams for one particular combination of strategies: "angle restriction" is applied, but "selection of stations" is not. First column: the frequency distribution histogram of all queries in the "snapshot" according to (a) the number of nodes met by the search (granularity: 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.

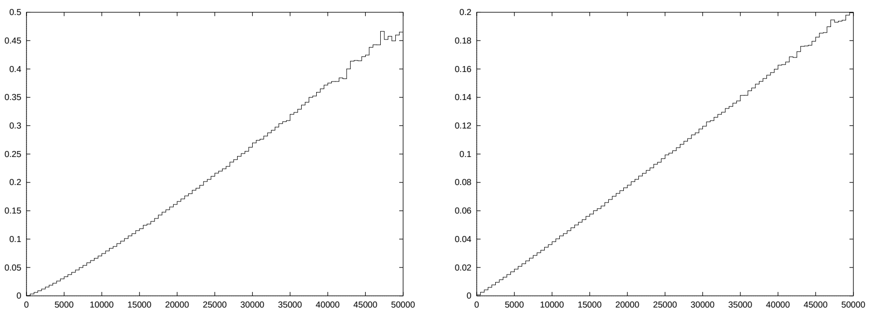

Fig. 6. The relation between the number of nodes hit by the search (abscissa) and the cpu time in seconds for the heap and the dial implementation, respectively (granularity: 500 nodes).

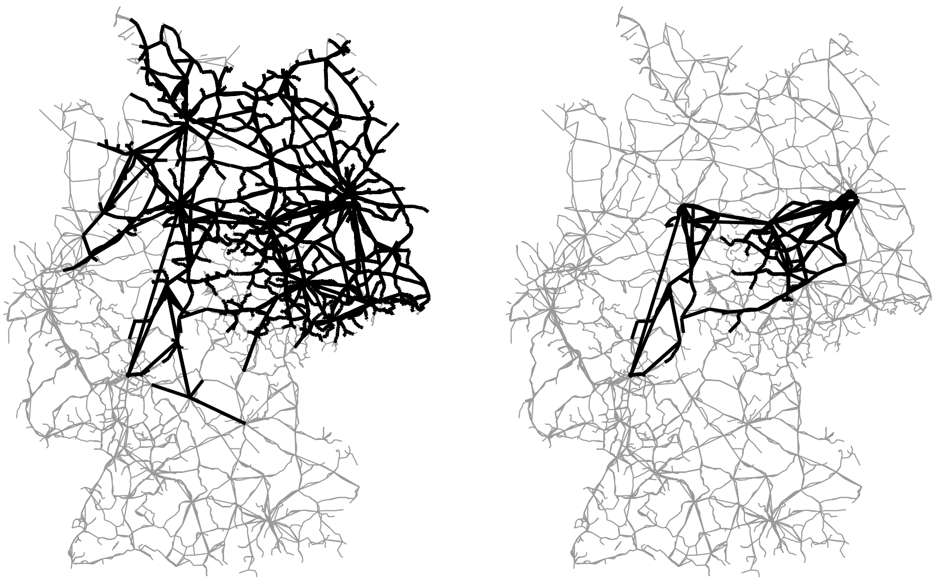

Fig. 7. The left picture shows the edges hit by Dijkstra's algorithm from Berlin Main East Station until the destination Frankfurt/Main Main Station is reached. In the right picture, the strategy "angle selection" was applied to the same query.

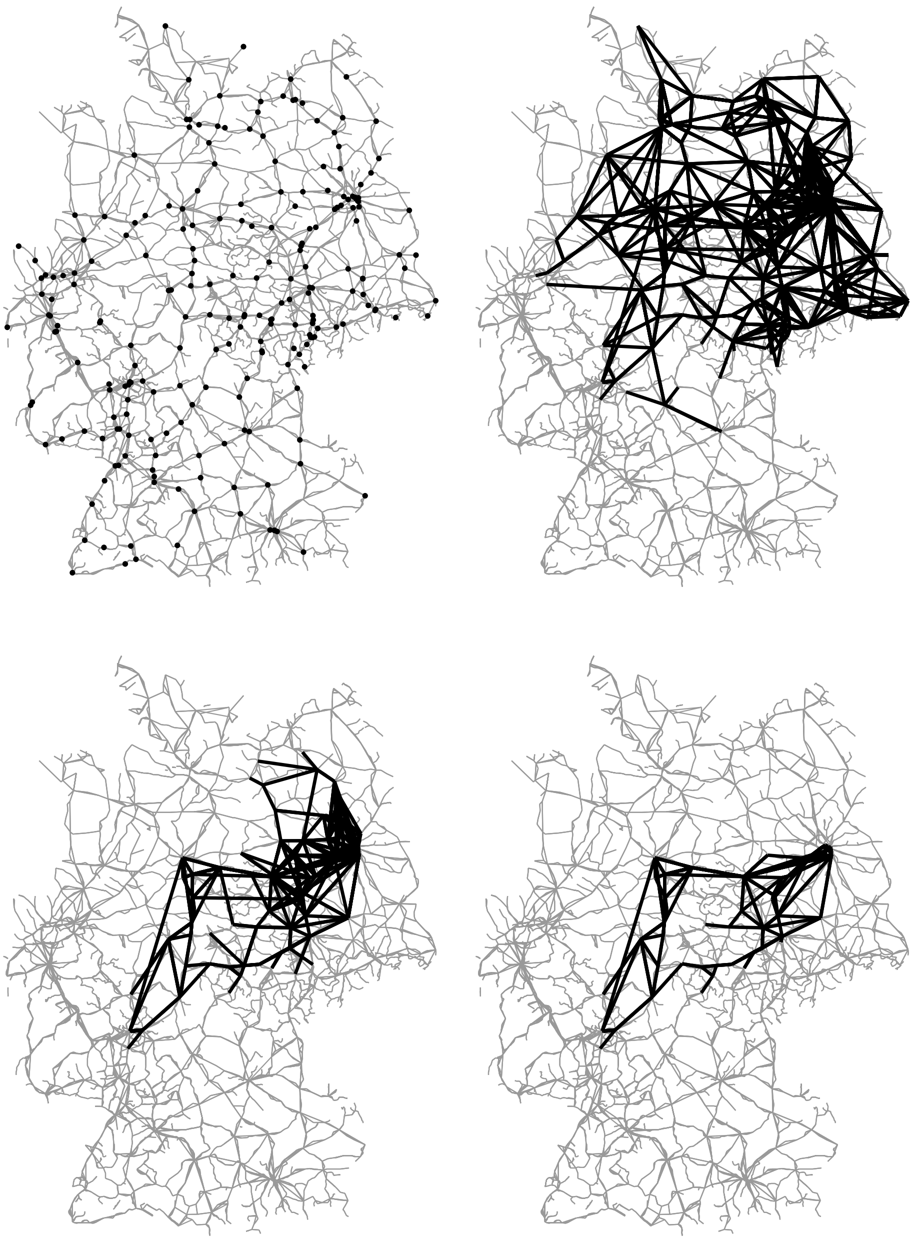

Fig. 8. The first picture shows the 225 stations selected for the study on strategy "selection of stations." The remaining three pictures refer to the same query as in Fig. 7. However, now the strategy "selection of stations" is applied with no angle restriction (upper right), with angles computed from the train graph (lower left), and with angles computed from the auxiliary graph itself. The train graph is shown in the background. The highlighted edges are the edges of the auxiliary graph hit by the search.