Experimental Study of Speed Up Techniques for Timetable Information Systems

Reinhard Bauer, Daniel Delling, and Dorothea Wagner
Karlsruhe Institute of Technology, Department of Computer Science, 76128 Karlsruhe, Germany

In recent years, many speed up techniques for DIJKSTRA Algorithm have been developed. Unfortunately, research mainly focused on road networks although fast algorithms are also needed for other applications like timetable information systems. Even worse, the adaption of recently developed techniques to graphs deriving from timetable information problems is often more complicated than expected. In this work, we check whether results from road networks are transferable to timetable information systems. To this end, we present an extensive experimental study of the most prominent speed up techniques on inputs deriving from different applications. It turns out that recently developed techniques are much slower on graphs derived from timetable information problems than on road networks. In addition, we gain interesting insights into the behavior of speed up techniques in general. © 2010 Wiley Periodicals, Inc. NETWORKS, Vol. 57(1), 38–52 2011

Keywords: graph algorithms; shortest paths; algorithm engineering; timetable information; experimental study

1. INTRODUCTION

Computing shortest paths in networks is used in many real-world applications like routing in road networks, timetable information, or airplane scheduling. In general, DIJKSTRA's algorithm [20] can be used to solve this problem. Unfortunately, the algorithm is too slow to be used on huge datasets, e.g., the US road network has more than 20 million nodes. In order to reduce query times for typical instances like road or railway networks, several speed up techniques have been developed during the last years (see [17] for an overview). Most recent research [3, 15] has made the calculation of distances within a road network a matter of microseconds.

Received January 2008; accepted October 2009
Correspondence to: D. Delling; e-mail: delling@ira.uka.de
Contract grant sponsor: Future and Emerging Technologies Unit of EC (IST Priority–6th FP); Contract grant number: FP6-021235-2 (Project ARRIVAL)
Contract grant sponsor: DFG; Contract grant number: WA 654/16-1
DOI 10.1002/net.20382
Published online 24 March 2010 in Wiley Online Library (wileyonlinelibrary.com).

Unfortunately, due to the availability of huge road networks, much of the recent research has focused only on such networks [3, 17, 45]. However, fast algorithms are needed for other applications as well. One might expect that all speed up techniques can simply be used in any other application, yet several problems arise. On the one hand, several assumptions which hold for road networks may not hold for other networks: e.g., in timetable information systems bidirectional search is prohibited as the arrival time is unknown in advance. Performance is the other big issue. The fastest methods [5] heavily exploit properties of road networks in order to gain their huge speed ups. Furthermore, most of the developed techniques only work in static scenarios, i.e., edge weights do not change between two requests. However, in railway networks, delays occur frequently. Thus, a solution for the dynamic timetable information problem is required.

In this work, we evaluate the most prominent speed up techniques on different types of input classes. One focus hereby lies in graphs deriving from timetable information systems. In such systems, we want to solve the earliest arrival problem (EAP) in a transportation network: Given two stations S and T, what is the earliest arrival at T if departing from S at time τ. In general, two approaches exist for solving the EAP: the time-dependent and time-expanded approaches. In the former model, each station is modeled by one node and an edge is inserted iff a direct connection between two stations exist. Several weights are assigned to each edge. Each weight represents the travel time of a train running from one station to another. In the time-expanded model, time-dependency is rolled out such that each edge represents a specific train running from one station to the other. While the time-dependent approach yields smaller inputs (and hence, smaller query times), the time-expanded approach allows a more flexible modeling of additional constraints.

At a glance, using speed up techniques developed for static (non time-dependent) road networks on time-expanded [39] graphs for timetable information systems seems promising. Since road networks seem to have similar properties as railway networks—both incorporate some kind of natural hierarchy and both are sparse—one might expect that speed up techniques yield the same performance as on road networks. However, our study reveals that speed up techniques perform significantly worse on time-expanded

graphs than on road networks. Even worse, obtained speed ups are below the expansion factor of ∼250 that exists between the time-dependent and time-expanded models [39]. As a consequence, running a simple time-dependent DIJKSTRA on the time-dependent graph is faster than any speed up techniques on the corresponding time-expanded graph. With the obtained results, we conclude that for pure performance issues the time-dependent model is somewhat superior to the time-expanded model. In addition, delays seem to be incorporated more easily in the time-dependent approach.

In addition, our extensive experimental study leads to intriguing insights into the behavior of speed up techniques. For small-world inputs, the biggest speed up is achieved by simply switching from uni- to bidirectional search and almost all speed up techniques do not yield an additional speed up. Moreover, we reveal the influence of density and diameter on the performance of the techniques. As most algorithms have only been tested on road networks, these new results are of independent interest.

1.1. Related Work

To the best of our knowledge, systematic experiments with different inputs and various speed up techniques for DIJKSTRA's algorithm can only be found in [29, 30]. However, in their work, the authors only use condensed railway networks and after its publication, several additional speed up techniques have been developed which we incorporate in this work. Moreover, most of the inputs used here are also evaluated in [4]. In [22, 23] additional tests—besides road networks—on grid graphs are performed. Moreover, work on parallel shortest path algorithms is evaluated on many other problem instances than road networks [2, 10, 34, 35]. The authors observe that the behavior of their algorithms also highly depends on the input.

There has been some research on adapting speed up techniques to timetable information applications. In [44] basic speed up techniques are used in time-dependent and time-expanded timetable information graphs. In [48], the multilevel speed up technique is applied to railway graphs. Geometric containers were evaluated in [49] on such graphs as well. However, to the best of our knowledge, no extensive tests incorporating all recently developed speed up techniques have been published yet.

A preliminary version of this work has been published [6]. However, here we incorporate a very recent speed up technique, namely SHARC [4]. It should be noted that based on the insights presented in this article we were able to develop further improvements, see e.g., [5, 7, 13, 16]. Moreover, this experimental study was the first one that systematically evaluated recent high-performance speed up techniques on inputs other than road networks.

1.2. Overview

This article is organized as follows. The most prominent speed up techniques are introduced in Section 2. In Section 3, we briefly discuss existing approaches for modeling timetable information problems as graphs. For all three approaches we discuss advantages and disadvantages with a focus on the effort of adapting speed up techniques to each model. Our extensive experimental study is located in Section 4, where we evaluate the speed up techniques from Section 2 on several real-world and synthetic datasets. Our work concludes with a summary in Section 5.

2. SPEED UP TECHNIQUES

Here, we briefly present DIJKSTRA's algorithm and the speed up techniques which are evaluated in Section 4 (for a more detailed overview see [17]). Because of the fact that many speed up techniques exist, we restrict ourselves to the most prominent ones and to those which do not need a layout of the input graph. For all techniques, we use the most sophisticated variant.

2.1. DIJKSTRA's Algorithm [20]

The classical algorithm for computing the shortest path from a given source s to all other nodes in a directed graph with non-negative edge weights is due to Dijkstra. The algorithm maintains a priority queue Q and a label for each node u depicting the tentative distance from the source s to node u. At each step, the algorithm removes (or scans) the node u from Q with minimum distance to s. Then, all outgoing edges (u, v) of u are relaxed, i.e., the algorithm checks whether the path via u to v is shorter than the path to v found so far. If the path via u is shorter, v is either inserted into the priority queue or its priority is decreased. The algorithm terminates when all nodes have been scanned. If we are only interested in computing a shortest path from s to a given target t, we may stop the search as soon as t has been scanned by DIJKSTRA's algorithm.

2.2. Bidirectional DIJKSTRA [11, 21]

The most straightforward speed up technique is bidirectional search. An additional search is started from the target node and the query stops as soon as both searches meet. The tuning parameter of this approach is the way forward and backward search are alternated. We here use a strategy that strictly alternates between both searches, balancing the work between them. Note that most sophisticated methods are bidirectional approaches.

2.3. ALT [21, 24]

Goal directed search, also called A^* [27], pushes the search toward a target by adding a potential to the priority of each node. Given a 2-dimensional layout, the usage of Euclidean potentials requires no preprocessing. The ALT algorithm, introduced in [21], obtains the potential from the distances to certain landmarks in the graph. Although this approach requires a preprocessing step, it is superior (compared to Euclidean potentials) with respect to search space

and query times. In this work, we use the latest variant of ALT, introduced in [24], with 16 maxCover landmarks as the representative of goal-directed search. The main advantage of ALT is its simple implementation and it can be used—without modification for most updates—in a dynamic and time-dependent scenario [19, 40], i.e., edge weights may change between two queries. The main downside of ALT are very fluctuating query times.

2.4. *Arc-Flags* [32,33,36]

This approach uses a pruning strategy, i.e., by attaching additional data to edges, a modified DIJKSTRA checks whether an edge can or cannot be on the shortest path to the target. More precisely, the Arc-Flag approach partitions the graph into cells and attaches a label to each edge. A label contains a flag for each cell indicating whether a shortest path to the corresponding cell exists that starts with this edge. As a result, a bidirectional Arc-Flags-DIJKSTRA often only visits those edges which lie on the shortest path of a long-range query. However, no speed up can be achieved for queries within a cell and the preprocessing effort is very high. In this work, we use the variant as described in [36].

Let (u, v) be an edge with both endpoints in the same cell C. Note that the arc-flag of (u, v) for the cell C is most often set. Because of these so called own-cell flags an Arc-Flags-DIJKSTRA yields no speed up for queries within the same cell. Even worse, using a unidirectional query, more and more edges become important when approaching the target cell (the coning effect) and finally, all edges are considered as soon as the search enters the target cell. While this coning effect can be weakened by a bidirectional query, the former also holds for such queries. Thus, a two-level approach is introduced in [37] which weakens these drawbacks as cells become quite small on the lower level. It is obvious that this approach can be extended to a multilevel approach.

2.5. *SHARC* [4]

It is an extension of the Arc-Flags approach. By using contraction—a routine that iteratively removes unimportant nodes and adds edges to preserve correct distances between remaining nodes—during preprocessing most of the disadvantages of Arc-Flags can be remedied. The result is a fast unidirectional query algorithm, which is especially advantageous in scenarios where bidirectional search is prohibitive, e.g., timetable information systems. In general, SHARC can be interpreted as a goal-directed technique that incorporates hierarchical aspects implicitly. In this work, we use the implementation of SHARC due to [4].

2.6. *Highway Hierarchies* [45]

This approach is a hierarchical method, i.e., an approach trying to exploit the hierarchy of a graph. Therefore, the network is contracted and then "important" edges—the highway edges—are identified. By rerunning those two steps, a natural hierarchy of the network is obtained. The contraction phase builds the core of a level and adds shortcuts to the graph. The identification of highway edges is done by local DIJKSTRA executions. In this work, we use the variant of Highway Hierarchies (HH) as described in [45]. This variant stops building the hierarchy at a certain point and computes a distance table containing all distances between the core-nodes of the highest level. The advantages of HH are very low preprocessing and query times (15 min of preprocessing on the Western European road network result in query times of 0.5 ms). However, this approach loses performance when using other metrics than travel times [18].

2.7. *RE/REAL* [22,23]

Reach [25] is a centrality measure based on the intuition that a node is important if it is situated in the middle of long shortest paths. In [25], reach is used as a node-label in order to prune the search. Some crucial disadvantages, e.g., preprocessing time, are remedied by enriching the graph by shortcuts in [22]. In addition, this approach naturally combines with ALT yielding high speed ups in road networks. The RE algorithm is a bidirectional reach-pruning DIJKSTRA on a shortcut-enriched graph, while REAL is the combination of RE and ALT. Note that RE can be interpreted as a hierarchical method. RE has similar advantages and disadvantages like HH, but preprocessing takes longer than for HH. The advantage of RE over HH is its sound combination with ALT, which cannot be combined with HH easily [18].

2.8. *Highway-Node Routing* [46]

This approach computes for a given sequence of node sets $V =: V_0 \supseteq V_1 \supseteq \cdots \supseteq V_L$ a hierarchy of overlay graphs [28, 47, 48]: the level-ℓ overlay graph consists of the node set V_ℓ and an edge set E_ℓ that ensures the property that all distances between nodes in V_ℓ are equal to the corresponding distances in the underlying graph $G_{\ell-1}$. A bidirectional query algorithm takes advantage of the multilevel overlay graph by never moving downwards in the hierarchy—by that means, the search space size is greatly reduced. A node classification provided by the Highway Hierarchies approach was used to define the highway-node sets leading to a multilevel overlay graph with about ten levels. Hence, the performance of Highway-Node Routing is very similar to pure Highway Hierarchies but the advantage of the former is its capability to efficiently update the preprocessing and low memory consumption. However, as performance is similar to Highway Hierarchies and we use static graphs in this work, we do not consider Highway-Node Routing.[1]

2.9. *Transit-Node Routing* [3]

When you start from a source node and drive to somewhere "far away," you will leave your current location via

[1] Preliminary experiments confirm that Highway-Node Routing performs very similar to Highway Hierarchies on most inputs.

one of only a few 'important' traffic junctions, called access nodes. Transit-Node Routing computes the distances between all such access nodes and between every node with its nearest access nodes in advance. Long-range queries can then be answered by a few table lookups. However, low-range queries still have to use another speed up technique to be handled efficiently. Transit Node Routing is tailored to road networks and it turns out that applying this technique in other scenarios is quite complicated: The number of transit nodes "explodes" on denser graphs and the underlying low-range speed up technique has to be adapted as well. As one of the most efficient variants of Transit Node Routing is based on Highway Hierarchies, the performance of Highway Hierarchies most likely reflects the performance of Transit Node Routing.[2] Hence, we do not use this technique in this work.

2.10. Example

Figure 1 shows the search space of some of the above-mentioned speed up techniques running the same query on a road network. A black edge depicts that it has been relaxed by the query algorithm. The shortest path is drawn thicker. Note that for SHARC and HNR shortcuts are inserted into the graph which we unpack for visualization. As a consequence, the search space may look bigger than for other techniques, but the number of relaxed edges may actually be smaller.

We observe that ALT gives the search an excellent sense of goal-direction but almost all nodes are visited near the source and the target of the query. Highway-Node Routing improves on the latter but is not goal-directed. SHARC yields the smallest search space by incorporating both hierarchical and goal-directed properties.

3. MODELING TIMETABLE INFORMATION SYSTEMS

In this section, we briefly present existing approaches to model (dynamic) timetable information systems as graphs (cf. [39] for details). In addition, we discuss problems of adapting speed up techniques to these models and how well delays can be covered.

3.1. Condensed Model

The easiest model is the condensed model. Here, a node is introduced for each station and an edge is inserted if a direct connection between two stations exists. The edge weight is set to be the minimum travel time over all possible connections between these two stations. See Figure 2 for an example. The advantage of this model is that the resulting graphs are quite small and we are able to use speed up techniques without modifications. Unfortunately, several drawbacks exist. First of all, this model does not incorporate the actual departure time from a given station. Even worse, travel times highly depend on the time of the day and the time needed for changing trains is also not covered by this approach. As a result, the calculated travel time between two arbitrary stations in such a graph is only a lower bound on the real travel time.

3.2. Time-Dependent Model

This model tries to remedy the disadvantages of the condensed model. The main idea is to use time-dependent edges. Hence, each station is also modeled by a single node and an edge is again inserted iff a direct connection between two stations exist. But unlike for the condensed model, several weights are assigned to each edge. Each weight represents the travel time of a train running from one station to another. The edge used during a query is then picked according to the departure time from the station. See Figure 3 for a small example. The advantage of this model is its small size and the obtained travel time is feasible. Furthermore, delays can easily be incorporated: the corresponding weight—representing the delayed connection—of an edge can simply be increased. However, adapting speed up techniques to time-dependent graphs is more complicated than expected. While for time-independent graphs speed ups of over three million can be achieved [5], the best results for time-dependent graphs only yield speed up factors of $\approx$215 [13]. In addition, this model does not cover transfer times, yet this can be remedied as shown in [43].

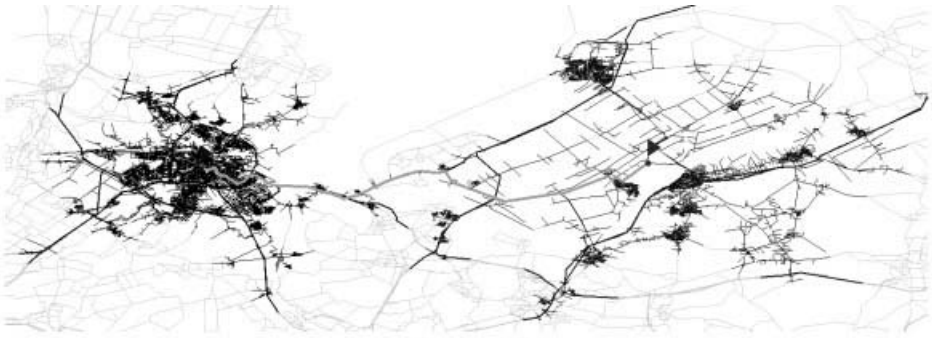
Note that the time-dependent model can be interpreted as an extension of the condensed model. In this work we evaluate speed up techniques on the condensed model in order to select techniques that are worth adapting to the dynamic time-dependent model.

3.3. Time-Expanded Model

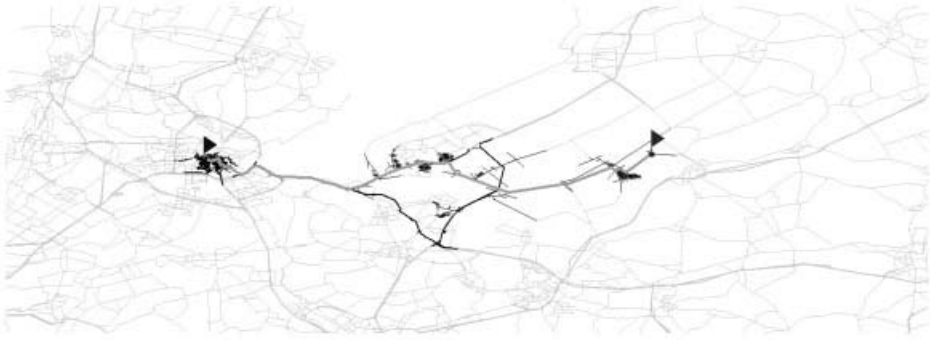
This is another model that does not rely on time-dependent edge weights and thus it is easier to use known speed up techniques in this model. Here, a node is used for each arrival and departure event. An edge is inserted for each connection between two events. Figure 4 gives a small example. The main downside of this approach is that the resulting graphs are much bigger than for the time-dependent approach. For our datasets, the number of nodes is roughly 250 times higher. Note that such graphs are strongly connected as timetables are periodic.

In general, most unidirectional speed up techniques can be used out-of-the-box on such a time-expanded graph. However, sophisticated methods gain their speed ups from a bidirectional search that needs to know the exact target node. Even worse, RE and HH only work correctly if used in a bidirectional manner. Unfortunately, in this model each node represents a specific event within the network and thus it is complicated to pick the target node from which to start the backward search. In addition, some unidirectional approaches, e.g., unidirectional ALT, also need the exact target node in order to work properly. Another pitfall originates from the model. The ordering of nodes within

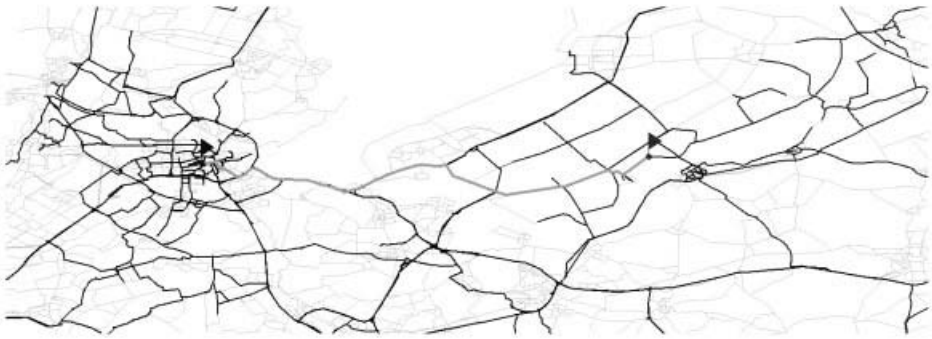
[2] However, with proper modifications, it seems possible to adapt Transit Node Routing to some of the inputs used.



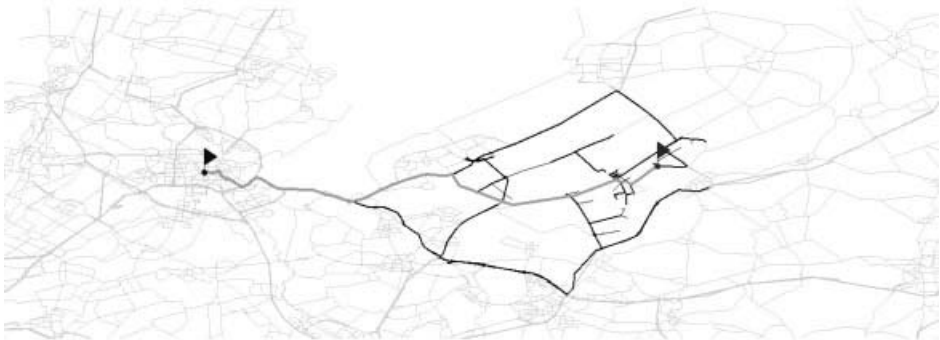
(a) Bidirectional Dijkstra



(b) Bidirectional ALT



(c) Highway-Node Routing



(d) SHARC

FIG. 1. Search space of some of the examined speed up techniques.

a station is very important for the correctness of timetable information queries. Whenever a delay occurs, trains may arrive in a different order than expected, leading to a complete change of the inner-edge structure of a station. As a consequence, delays yield changes in the topology within the network which results in a larger effort of updating the preprocessed data of the speed up techniques. Thus, adapting techniques to a dynamic time-expanded model appears to be very complicated [38].

Note that transfer times are not covered correctly. For this reason, this model is called the simple time-expanded model. However, this can be remedied by an extended model, but

FIG. 2. Condensed network of our European timetable information data, provided by HaCon [26] for scientific use.

the graph size additionally increases by a factor of ~ 2. In this work, we evaluate the speed up techniques on the static simple time-expanded model in order to pick the most promising technique that is worth adapting to the dynamic extended time-expanded model.

4. EXPERIMENTS

In this section, we present an extensive experimental evaluation of the speed up techniques on different types of graphs. Our implementation is written in C++ (using the STL at some points). As priority queue we use a binary heap. Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.1. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2×1 MB of L2 cache. The program was compiled with GCC 4.1, using optimization level 3.

Default Settings: Unless otherwise stated, we use the following settings. For ALT, we use 16 maxCover [24] landmarks. In our Arc-Flags setup, we use 128 cells obtained from METIS [31]. For SHARC, we apply a 2-level partitioning—obtained from SCOTCH [41]—with 112 cells on the upper level and 16 cells per supercell on the lower level (cf. [4] for details). In addition, we evaluate the hierarchical RE algorithm [22] and Highway Hierarchies (HH) [45]. The performance of both approaches highly depends on the chosen preprocessing parameters which we here tune manually. For HH, we use a distance table as soon as the contracted graph has fewer than 10,000 nodes. Moreover, we evaluate the combination of RE and ALT, named REAL, without reach-aware landmarks [23].

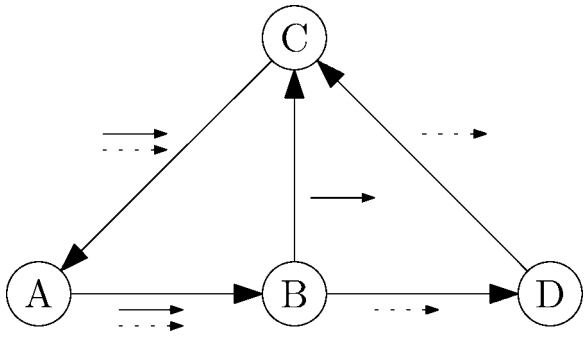


FIG. 3. Time-dependent model. We have four stations and two train routes. The first runs from A to B to C and back to A, while the route of the second one is A→B→D→C→A.

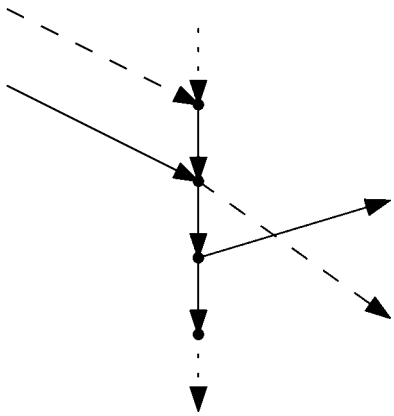


FIG. 4. Time-expanded model. The figure shows parts of a station with two trains arriving and departing.

TABLE 1. Performance of the speed up techniques on the condensed railway network of Europe.

	Travel time				Distance				Unit				Random			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0.0	0	14 761	3.48	0.0	0	14 603	2.82	0.0	0	14 691	3.35	0.0	0	14 549	5.26
BiDijkstra	0.0	0	7 520	1.83	0.0	0	8 615	1.69	0.0	0	1 158	0.27	0.0	0	1 515	0.53
uni ALT	0.1	128	1 191	0.47	0.1	128	1 007	0.37	0.1	128	1 840	0.90	0.1	128	1 835	1.00
ALT	0.1	128	348	0.21	0.1	128	374	0.21	0.1	128	109	0.10	0.1	128	108	0.10
uni Arc-Flags	0.6	47	236	0.13	0.5	47	327	0.14	0.6	47	160	0.08	0.7	47	178	0.13
Arc-Flags	1.1	94	50	0.03	1.0	94	75	0.03	1.1	94	19	0.01	1.5	94	26	0.02
RE	0.1	27	272	0.13	0.1	20	258	0.12	0.1	16	377	0.15	0.8	22	739	0.35
uni REAL	0.2	155	116	0.12	0.2	148	87	0.09	0.2	144	687	0.64	0.9	150	751	0.76
REAL	0.2	155	72	0.08	0.2	148	70	0.07	0.2	144	66	0.09	0.9	150	81	0.14
HH	0.1	46	88	0.04	0.1	78	226	0.11	0.1	24	338	0.12	0.1	38	125	0.06
SHARC	0.1	14	168	0.03	0.1	16	175	0.03	0.1	12	161	0.03	0.3	2	159	0.03

Because of the graph size, we use the distance table for HH as soon as the core has less than 1,000 nodes. Column $[B/n]$ denotes the preprocessing space in bytes per node.

Methodology: In the following we report preprocessing effort and query performance of all speed up techniques. For the former, we report the preprocessing time and the resulting additional space (in bytes per node), while for the latter we report the average number of scanned nodes, i.e., the number of nodes taken from the priority queues, and the resulting query times. The values for query-performance are obtained from running 10,000 random queries, i.e., the source s and the target t are picked uniformly at random. Note that all figures in this article refer to the scenario that only the lengths of the shortest paths have to be determined, without outputting a complete description of the paths.

4.1. Timetable Information

4.1.1. Condensed Model. We start our experimental study with the condensed network of Europe (cf. Fig. 2), based on timetable information data provided by HaCon [26] for scientific use. The graph has 29,578 nodes and 86,566 edges. To check whether the speed ups derive from the topology of the network or if they are due to the metric used we consider—besides travel time—three additional metrics: distance depicts the real distance between two stations, unit assigns a weight of 1 to each edge, and random reassigns to each edge a weight between 1 and 1000 picked uniformly at random. Computational results are shown in Table 1.

We observe that plain DIJKSTRA scans the same number of nodes independent of the applied metric. However, query times vary: DIJKSTRA is two times faster on the distance metric than on the random one. The number of DECREASEKEY operations causes these different running times. Surprisingly, switching to bidirectional DIJKSTRA has a completely different impact for different metrics. While for travel times and distances, a speed up of factor 2 is observed, queries using the unit metric run 12 times faster. As shown in Figure 2, several direct connections within the network exist: setting the weight of these edges to 1 drastically reduces the search space of bidirectional DIJKSTRA as the forward and backward search meet earlier. This observation also holds somewhat more weakly for the random metric; here the speed up factor is 10.

Analyzing our speed up techniques, all approaches are able to preprocess the graph in less than 1 min. The fastest technique is bidirectional Arc-Flags having query times of below 30 μs for all metrics. As for bidirectional DIJKSTRA, the lowest query times are achieved for the unit metric which is again due to direct connections. SHARC requires the lowest amount of additional memory and thus has the best combination of query times and preprocessing. Nevertheless, as we use the condensed model, the obtained travel times cannot be used in a real-world environment (cf. Section 3).

4.1.2. Time-Expanded Model. Our second set of experiments is executed on three simple time-expanded graphs (cf. Section 3). The first shows the local traffic of Berlin/Brandenburg, with 2,599,953 nodes and 3,899,807 edges, the second one represents local traffic of the Ruhrgebiet (2,277,812 nodes, 3,416,597 edges), and the last graph depicts long distance connections of Europe (1,192,736 nodes, 1,789,088 edges). Table 2 gives an overview of the performance of our speed up techniques on these instances.

Note that RE, ALT, and HH cannot be used out-of-the-box for time-expanded networks (cf. Section 3). To gain insights into the performance of these techniques, we also use bidirectional speed up techniques by picking a random event at the target station. Thus, these bidirectional experiments are intended to give hints whether it is worth focusing on adapting bidirectional search to such graphs. Only SHARC and unidirectional Arc-Flags—with a partitioning by station—are applicable. Unidirectional Arc-Flags performs roughly 6 times faster than unidirectional DIJKSTRA. This is much less than the bidirectional variant of Arc-Flags (speed up factor of 45-65). Thus, it may be worth focusing on the question of how to use bidirectional search in this scenario. However, we observe very long preprocessing times for Arc-Flags on these

TABLE 2. Performance of the speed up techniques on time-expanded railway networks.

	Berlin/Brandenburg				Ruhrgebiet				Long distance			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	1 299 830	406.2	0	0	1 134 420	389.2	0	0	609 352	221.2
BiDijkstra	0	0	496 281	151.3	0	0	389 577	122.8	0	0	143 613	43.8
uni ALT	10	128	383 921	133.6	10	128	171 760	64.7	5	128	71 194	26.0
ALT	10	128	47 764	22.9	10	128	59 516	30.5	5	128	31 367	15.0
uni Arc-Flags	2 240	24	172 362	72.2	2 323	24	158 174	66.4	1 008	24	74 737	32.4
Arc-Flags	4 479	48	24 004	9.2	4 646	48	28 448	10.7	2 016	48	10 560	3.5
RE	182	39	27 095	25.5	290	45	38 397	39.8	63	43	8 978	8.3
uni REAL	192	167	20 062	22.2	300	173	16 649	21.1	68	171	6 335	8.8
REAL	192	167	4 159	6.6	300	173	7 867	13.3	68	171	2 479	4.5
HH	38	263	5 285	56.1	65	202	9 528	196.2	12	386	1 930	7.3
SHARC	602	9	11 006	3.8	615	8	12 412	4.2	209	15	7 519	2.2

networks. The situation changes for SHARC; here the preprocessing times are reasonable and the query performance is the best of all applied speed up techniques. Although other approaches have a smaller search space, e.g., REAL, the smaller computational overhead of SHARC yields smaller query times. However, only ALT and HH can preprocess all graphs in less than 1 hour. RE seems to have problems on the local traffic networks as preprocessing takes longer than 3 hours and the achieved speed ups are only modest, while this does not hold for long distance connections. Regarding query times, HH has also problems with both local traffic networks: on Berlin/Brandenburg, HH is only 3 times faster than bidirectional DIJKSTRA, and on the Ruhrgebiet, HH is even slower. The problems of RE/HH derive from a weaker hierarchy within the local networks compared to the long distance graph. Local traffic networks do not incorporate high-speed trains while the latter do.

Summarizing, the fastest technique, SHARC, yields quite good speed up factors of around 100. However, the expansion of time-expanded graphs by a factor of 250 over the condensed—and hence also time-dependent—graphs cannot be compensated. Plain DIJKSTRA on a corresponding condensed network would be faster—with respect to query times—than any other speed up technique on the time-expanded model. Note that our input from Table 1 covers even more stations than any input from Table 2. Also note that plain DIJKSTRA can be used in a dynamic time-dependent scenario [9], and time-dependent ALT achieves an additional speed up factor of 5 over plain DIJKSTRA[19]. Moreover, SHARC can be used in time-dependent scenarios as well.

4.2. Road Networks

Like railway networks, road graphs incorporate some kind of hierarchy. Hence, one might expect that the speed up techniques have a similar performance on those two types of networks. We evaluate the German road network, provided by PTV AG [42] for scientific use. It has 4,377,307 nodes and 10,667,837 edges. We use four different metrics: travel time, distance, unit (each edge has length 1), and random. The latter reassigns edge weights uniformly at random from 1 to 1,000. We want to test whether the speed up techniques rely on the topology of the network or the speed up derives from the applied metric. Results can be found in Table 3.

TABLE 3. Performance of the speed up techniques on the German road graph using different metrics.

	Travel time				Distance				Unit				Random			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	2 214 820	1078.2	0	0	2 159 310	625.8	0	0	2 197 080	922.5	0	0	2 256 530	1335.4
BiDijkstra	0	0	1 210 570	545.0	0	0	1 428 140	405.7	0	0	952 405	378.4	0	0	1 006 260	530.0
uni ALT	23	128	139 121	51.2	18	128	95 385	33.8	19	128	135 728	48.5	23	128	143 551	59.4
ALT	23	128	22 150	12.4	18	128	45 496	23.1	19	128	16 329	8.0	23	128	21 433	12.2
uni Arc-F.	976	39	24 290	10.6	720	39	59 094	24.2	820	39	23 119	9.7	1 139	39	24 509	14.0
Arc-Flags	1 952	78	1 092	0.5	1 440	78	13 038	5.4	1 640	78	816	0.3	2 278	78	897	0.4
RE	18	22	5 080	3.1	20	27	10 666	9.4	16	19	4 210	2.6	20	30	4 879	3.5
uni REAL	41	150	1 804	1.8	38	155	1 642	2.1	37	147	2 210	2.2	43	158	2 369	2.7
REAL	41	150	1 035	1.2	38	155	1 556	2.3	37	147	978	1.1	43	158	1 130	1.4
HH	4	99	682	0.5	9	122	3 602	3.8	5	83	965	0.8	5	83	1 039	0.9
SHARC	16	13	1896	0.5	12	17	3 824	1.3	15	15	1 897	0.5	16	13	1 972	0.5

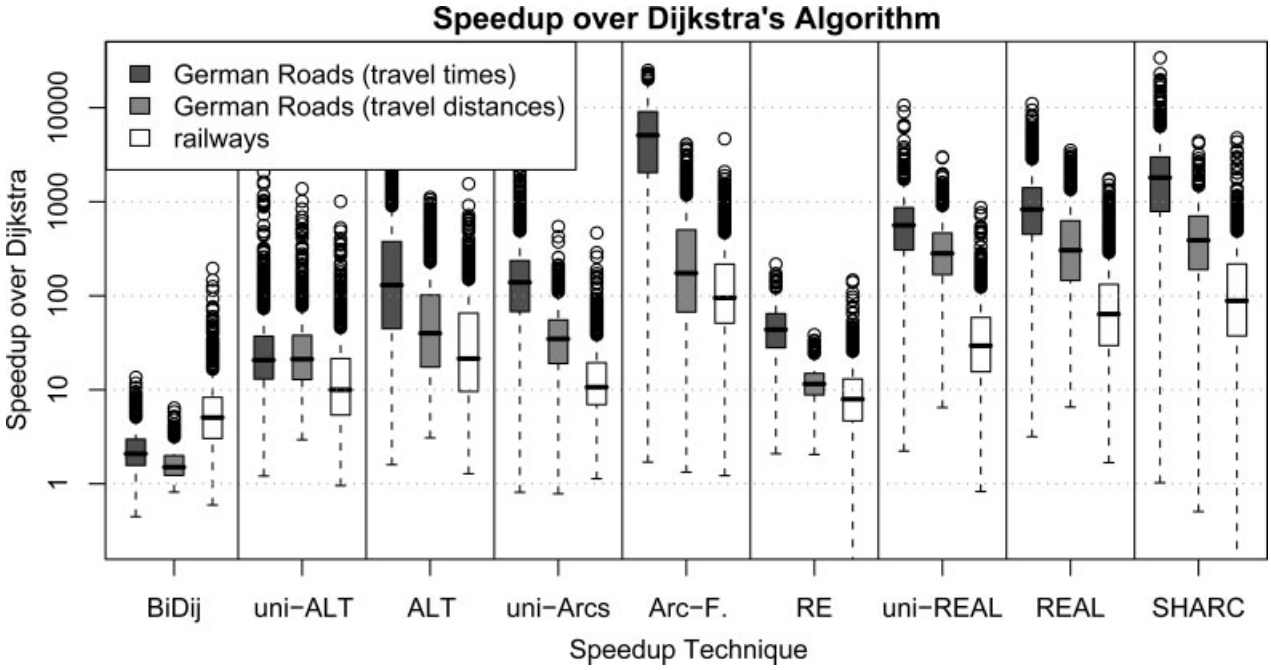


FIG. 5. Speed ups of various speed up techniques over DIJKSTRA's algorithm.

As expected, plain DIJKSTRA scans the same number of nodes for each metric. Stunningly, query times vary significantly when switching metrics: DIJKSTRA's algorithm is two times faster on the distance metric than on the random. This derives from the number of DECREASEKEY operations of the used priority queue. However, when switching from uni- to bidirectional DIJKSTRA, the situation changes. Similar to the other inputs evaluated so far (cf. Tables 1 and 2), the number of scanned nodes is not the same for each metric. In this scenario, the reason for this are the motorways which are favored differently by each metric resulting in a different termination of the bidirectional query algorithm.

Analyzing the speed up techniques, we observe very high preprocessing times for Arc-Flags which is due to the large number of DIJKSTRA executions during preprocessing, while HH can preprocess the complete German network much faster than any other technique. This result is not very surprising since HH was tuned for road networks and exploits properties of the (European) datasets. For example, curves on motorways are often modeled by a path with many degree-2 nodes which are shortcut during the preprocessing of HH. The same holds for RE. However, by adding contraction to Arc-Flags, i.e., SHARC, we observe that SHARC outperforms any hierarchical technique. Although bidirectional Arc-Flags yields better query times on the unit and distance metrics, preprocessing times of SHARC are much better. Moreover, recall that SHARC is a unidirectional technique. For ALT, we observe that the number of scanned nodes is almost the same for the travel time, unit, and random metrics. This holds for both the uni- and bidirectional variants. However, for the distance metric, the situation is different: the unidirectional variant is faster on this metric than on the others. On the contrary, the bidirectional variant loses performance when switching to the distance metric. This might be a reason for the surprising performance of REAL (the combination of RE and ALT) on the distance metric: the undirectional variant is faster than the bidirectional one (2.1 ms vs. 2.3 ms).

Summarizing, SHARC seems to be the best choice for road networks. Although HH yields faster preprocessing times, query performance of SHARC is better. Only when applying the unit or random metric, bidirectional Arc-Flags outperforms SHARC but at a price of much higher preprocessing times.

4.2.1. Similarity to Railway Networks. Comparing Tables 2 and 3 (for details, see Fig. 5) we observe that the speed up techniques perform much worse on time-expanded graphs than on road networks. So, at least for the time-expanded model the assumption of similar properties does not seem to hold. However, comparing Tables 1 and 3, and taking the difference in size into account, it seems as if road networks can be used as an alternative for condensed railway networks. But as the graph sizes are very different from each other, we perform another test on a road network of similar size to the European railway network. We choose the road network of Luxemburg which has 30,746 nodes and 71,655 edges. Again, we use the four metrics of travel time, distance, unit, and random. Results can be found in Table 4.

We observe that for the most important—at least in our application—metric, i.e., travel time, the performance of all speed up techniques is very similar to their performance on the condensed railway network. Differences in the unit and random metrics derive from direct connections within the railway network that do not exist in road networks. We conclude that road networks can be used as alternative data sources for the condensed model if timetable data is lacking.

TABLE 4. Performance of the speed up techniques on the Luxemburg road network.

	Travel time				Distance				Unit				Random			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0.0	0	15 293	3.12	0.0	0	15 230	2.87	0.0	0	15 441	2.69	0.0	0	15 156	3.60
BiDijkstra	0.0	0	7 691	1.63	0.0	0	9 526	1.77	0.0	0	7 304	1.28	0.0	0	7 056	1.63
uni ALT	0.1	128	1 375	0.53	0.1	128	1 052	0.37	0.1	128	1 099	0.41	0.1	128	1 122	0.47
ALT	0.1	128	448	0.21	0.1	128	451	0.21	0.1	128	458	0.21	0.1	128	456	0.23
uni Arc-Flags	0.3	37	470	0.17	0.3	37	614	0.23	0.3	37	421	0.15	0.4	37	435	0.22
Arc-Flags	0.7	74	178	0.06	0.6	74	250	0.09	0.6	74	133	0.05	0.8	74	144	0.07
RE	0.1	28	532	0.21	0.1	29	348	0.16	0.1	22	358	0.12	0.1	34	385	0.16
uni REAL	0.2	156	229	0.20	0.2	157	105	0.10	0.2	150	171	0.14	0.2	162	174	0.16
REAL	0.2	156	119	0.11	0.2	157	86	0.09	0.2	150	97	0.08	0.2	162	101	0.10
HH	0.1	219	91	0.05	0.1	140	241	0.12	0.1	69	299	0.14	0.1	204	111	0.06
SHARC	0.1	12	184	0.03	0.1	15	213	0.04	0.1	15	185	0.03	0.1	12	185	0.03

Because of the graph size, we use the distance table for HH as soon as the core has less than 1,000 nodes.

4.2.2. Important Subgraphs. The European road network includes roads which are closed to public traffic, e.g., pedestrian zones, etc. By removing these roads from the German network, the number of nodes decreases to 3,523,370 and the number of edges to 8,133,531, respectively. As these roads seem unimportant to shortest path computations, one might expect that the performance of the evaluated speed up techniques hardly changes if they are included or not. In addition, degree-1 and degree-2 nodes seem to be unimportant for shortest paths as well: nodes with degree 1 can only be starting or ending points of a route and degree-2 nodes can often be shortcut. Table 5 shows the results of all speed up techniques for excluded non-public roads, using the 2-core as input (3,183,701 nodes, 8,280,625 edges), the graph with shortcut degree-2 nodes (3,723,319 nodes, 9,363,584 edges), and the 2-core with shortcut degree-2 nodes (1,828,995 nodes, 5,469,750 edges). We use travel time as the metric.

Comparing the results from Tables 3 (travel time) and 5, we observe that the search space of uni- and bidirectional DIJKSTRA decreases with the size of the subgraphs. Astonishingly, this does not hold for query times: shortcutting degree-2 nodes yields higher query times than using the 2-core as input. The reason for this is that the numbers of edges differ: the 2-core has fewer edges than the other subgraph. However, this fact has no influence on bidirectional ALT. The algorithm has the same performance on the first three subgraphs and surprisingly, the performance is almost the same as on the full graph. Only when using the shortcut 2-core do query times decrease, which is mostly due to the graph's size.

The most interesting behavior is that of HH. On each subgraph the performance is almost the same as on the full graph. Recalling the way the hierarchy is built, the reason becomes obvious. Preprocessing of HH starts with a contraction step which consists of building the 2-core and shortcutting degree-2 nodes. Thus, HH has no advantage when applying these steps before preprocessing.

TABLE 5. Performance of the speed up techniques on the German road graph using different subgraphs.

	Only public				No deg. 2				2-Core				2-Core + no deg. 2			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	1 729 390	658.8	0	0	1 809 350	855.9	0	0	1 580 610	687.9	0	0	913 476	367.7
BiDijkstra	0	0	974 453	343.8	0	0	978 311	426.7	0	0	855 943	347.5	0	0	497 760	190.4
uni ALT	14	128	112 814	38.6	17	128	119 778	46.0	14	128	106 668	40.9	8	128	59 907	24.1
ALT	14	128	21 914	11.3	17	128	19 589	11.1	14	128	19 757	11.1	8	128	10 668	5.9
uni Arc-F.	610	37	20 583	8.3	794	40	19 683	8.4	638	42	19 655	8.4	335	48	11 755	5.2
Arc-Flags	1 220	74	1 067	0.4	1 588	80	710	0.3	1 276	83	1 038	0.4	670	96	618	0.3
RE	6	18	2 328	1.3	17	22	5 139	3.1	14	27	4 764	2.9	12	31	4 958	3.1
uni REAL	20	146	855	0.9	34	150	1 838	1.9	28	155	1 652	1.7	20	159	1 500	1.6
REAL	20	146	506	0.6	34	150	1 105	1.3	28	155	950	1.1	20	159	856	1.0
HH	2	45	660	0.5	4	115	679	0.5	4	128	677	0.5	4	207	661	0.5
SHARC	5	12	1 553	0.4	14	13	1 659	0.5	13	15	1 530	0.4	10	17	1 031	0.3

TABLE 6. Performance of the speed up techniques on different small-world graphs.

	Router				Citations				Coauthorship			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	94 717	89.0	0	0	134 136	190.8	0	0	153 885	125.5
BiDijkstra	0	0	216	0.3	0	0	742	1.5	0	0	320	0.4
uni ALT	2	128	23 430	36.8	2	128	28 853	68.6	2	128	38 173	51.5
ALT	2	128	320	1.7	2	128	850	4.7	2	128	667	2.2
uni Arc-Flags	351	102	5 453	12.9	1 488	138	46 318	113.7	507	105	28 225	62.8
Arc-Flags	702	204	42	0.1	2 977	276	231	0.7	1 014	209	117	0.3
RE	174	11	820	1.7	1 922	18	3 465	8.4	417	10	445	0.9
uni REAL	176	139	22 493	44.2	1 924	146	27 898	90.3	419	138	34 163	67.5
REAL	176	139	337	2.3	1 924	146	762	6.0	419	138	522	2.9
HH	38	1815	20 488	1 307.7	862	532	89 696	928.9	246	2982	61 703	1 713.7
SHARC	70	55	21 701	27.2	1 058	71	80 510	123.6	362	26	45 099	69.2

4.3. Other Inputs

To gain further insights into the behavior of speed up techniques, our last testsets use data that is completely different from transportation networks. On the one hand, we test the performance of speed up techniques on small-world graphs and on the other hand, we want to evaluate the influence of density and diameter of the input on the performance of speed up techniques. For our density testset we use so called unit-disc graphs with different average degrees. Our diameter testset uses multi-dimensional grid graphs with different numbers of dimensions as inputs.

4.3.1. Small World. Up to this point, we concentrated on graphs with some kind of hierarchy. In this test, we use small-world graphs as input without such a property. The first dataset represents the internet on the router level, i.e., nodes are routers and edges represent connections between routers. The network is taken from the CAIDA webpage [8] and has 190,914 nodes and 1,215,220 edges. The second graph is a citation network, i.e., nodes are papers and edges depict whether one paper cites another one. It is obtained from crawling the literature database DBLP [12] and has 268,495 nodes and 2,313,294 edges. The final dataset is a co-authorship [1] network (299,067 nodes and 1,955,352 edges) which is also obtained from the DBLP: nodes represent authors and two authors are connected by an edge if they coauthored a paper. Note that all edges of these inputs are weighted by 1. The results for these data are shown in Table 6.

The most interesting observation is that the biggest speed up is achieved by simply switching from uni- to bidirectional DIJKSTRA. This derives from the very small diameters of the graphs (less than 8 for all instances). Stunningly, only Arc-Flags yields an additional, while mild speed up. Taking the huge preprocessing time of more than 10 hours into account, the usage of Arc-Flags cannot be justified. Any other approach is even slower than bidirectional DIJKSTRA which is mainly due to computational overhead. Analyzing HH, this approach seems to have serious problems with small-world graphs. The reason is the stopping criterion (cf. [45]). Normally, bidirectional search can be stopped as soon as both search spaces meet. But for HH, this does not hold: the search has to be continued until both searches have reached the highest core or when the forward search scans a node which has a distance label greater than the shortest path seen so far.

We conclude that—as long as a bidirectional approach is allowed— applying a speed up technique with preprocessing is not worth the effort since plain bidirectional DIJKSTRA performs well enough. However, the situation changes if a scenario arises where bidirectional approaches are prohibited. In such a situation, unidirectional ALT yields a moderate speed up combined with a reasonable preprocessing effort.

4.3.2. Sensor Networks. Recently, the field of sensor networks has drawn wide attention. At a glance, routing in such networks has properties similar to routing in road networks. Thus, we evaluate so called unit-disc graphs which are widely used for experimental evaluations in that field. Such graphs are obtained by arranging nodes in the plane and connecting nodes with a distance below a given threshold. It is obvious that the density can be varied by applying different threshold values. In our study, we use graphs with $\sim$1,000,000 nodes and an average degree of 5, 7, and 10, respectively. As metric, we use the Euclidean distance between nodes according to their embedding. The results can be found in Table 7.

Uni- and bidirectional DIJKSTRA scan roughly the same number of nodes independent of the average degree but query times again increase with higher density due to more relaxed edges. Analyzing ALT, the bidirectional variant is twice as fast as the unidirectional algorithm on the instance with degree 5 while with degree 10, both approaches are equal to each other with respect to query times. The decreasing search space of unidirectional ALT is due to the increasing number of edges. With more edges, the shortest path is very close to the flight distance between source and target. In such instances, the potentials deriving from landmarks are very good. Arc-Flags yields very good query times but again at the price of high preprocessing times. Hierarchical methods

TABLE 7. Performance of speed up techniques on unit-disc graphs with different average degrees.

	Average deg. 5				Average deg. 7				Average deg. 10			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	487 818	257.3	0	0	521 874	330.1	0	0	502 683	399.0
BiDijkstra	0	0	299 077	164.4	0	0	340 801	225.1	0	0	325 803	269.4
uni ALT	8	128	22 476	17.1	8	128	16 634	15.1	10	128	14 561	16.0
ALT	8	128	9 222	8.5	8	128	10 565	11.8	10	128	11 749	15.6
uni Arc-Flags	53	80	8 556	7.9	299	112	16 445	16.8	801	160	21 413	24.2
Arc-Flags	105	160	2 091	1.8	598	224	4 761	4.6	1 602	320	7 019	7.5
RE	4	20	848	0.5	46	42	13 783	14.3	1 153	54	83 826	104.5
uni REAL	12	148	307	0.4	54	170	2 072	3.2	1 163	182	8 780	13.6
REAL	12	148	291	0.4	54	170	2 394	4.1	1 163	182	11 449	21.7
HH	2	251	203	0.2	12	549	5 068	8.5	71	690	23 756	49.1
SHARC	1	16	568	0.3	10	42	1 835	1.0	70	96	4 972	3.6

work very well for average degrees of 5 and 7. For a degree of 10, preprocessing and query times increase drastically. For RE, a reason is that node-labels are used for pruning the search. With increasing density, many edges are never used by any shortest path. Hence, query times increase as these edges cannot be pruned by using node-labels. SHARC outperforms any technique with less space and very small preprocessing times. However, the gap in query performance between ALT and SHARC gets smaller the denser the graph gets. Summarizing, SHARC performs best on these inputs. Only for high densities, ALT yields lower preprocessing times but still, SHARC yields a better query performance.

4.3.3. Grid Graphs. Our last testset explores the influence of graph diameter on the performance. Here, we vary the diameter of a graph by using multi-dimensional grid graphs with 2, 3, and 4 dimensions. The number of nodes is set to 250,000, and thus, the number of edges are 1, 1.5, and 2 million, respectively. Edge weights are picked uniformly at random from 1 to 1,000. These results can be found in Table 8.

As for sensor networks, unidirectional DIJKSTRA scans the same number of nodes on all graphs. However, due to more relaxed edges the query times increase with an increasing number of dimensions. As the diameter shrinks with increasing number of dimensions, bidirectional DIJKSTRA scans fewer nodes on 4-dimensional grids than on 2-dimensional grids. We already observed this effect more drastically for small-world graphs (cf. Table 6). Uni- and bidirectional ALT yield good speed ups combined with a low preprocessing effort. Our hierarchical representatives RE/HH perform very well on 2-dimensional grids but significantly lose performance at higher dimensions. The main reason is that the contraction phase of the algorithms fails.

Summarizing, ALT has the best trade-off with respect to preprocessing and query times on 3- and 4-dimensional grids. Only Arc-Flags and SHARC are faster but at the price of a much higher effort in preprocessing. Hierarchical methods like RE/HH can only compete with ALT on 2-dimensional grids.

TABLE 8. Performance of speed up techniques on grid graphs with different numbers of dimensions.

	2-Dimensional				3-Dimensional				4-Dimensional			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)	(min)	$[B/n]$	#scanned	(ms)
Dijkstra	0	0	125 675	36.7	0	0	125 398	78.6	0	0	122 796	137.5
BiDijkstra	0	0	79 962	24.2	0	0	45 269	28.2	0	0	21 763	20.3
uni ALT	1	128	5 452	2.5	2	128	4 223	3.8	3	128	5 031	7.5
ALT	1	128	2 381	1.5	2	128	1 807	2.2	3	128	1 329	2.5
uni Arc-Flags	45	64	4 476	1.9	415	94	8 996	5.7	1 559	122	25 125	26.8
Arc-Flags	89	128	1 340	0.6	830	189	1 685	1.0	3 117	244	2 800	2.3
RE	13	31	3 797	2.1	220	102	18 177	27.1	2 243	89	20 587	40.2
uni REAL	14	159	799	0.8	222	230	5 081	10.6	2 246	217	10 740	30.3
REAL	14	159	829	0.9	222	230	3 325	8.5	2 246	217	3 250	11.6
HH	2	1682	583	0.6	32	1954	17 243	95.8	680	662	61 715	343.0
SHARC	32	60	1 089	0.4	62	97	5 839	1.9	292	13	20 115	11.5

5. CONCLUSION AND OUTLOOK

We learned a lot about the performance of the most prominent speed up techniques on graph classes other than road networks. For graphs deriving from timetable information systems, the achieved speed up on time-expanded graphs is much smaller than the speed up achieved on road networks, even without the necessary modifications that will most probably decrease performance even further. In addition, the speed up obtained by all techniques is below the expansion factor of approximately 250 between time-dependent and corresponding time-expanded graphs. We observed that plain DIJKSTRA yields lower query times on a condensed network than any other speed up techniques on the time-expanded graphs. Recall that the time-dependent model can be interpreted as an extension of the condensed one. In [9], it is shown that plain DIJKSTRA can be used in a dynamic time-dependent scenario easily, and time-dependent ALT achieves an additional speed up factor of 5 compared to plain DIJKSTRA [19]. In addition, incorporating delays seems to be easier in the time-dependent model than in the time-expanded one [14, 38]. We conclude that it is promising to work on the dynamic time-dependent model for solving the timetable information problem.

Regarding time-expanded data, we observe that the often stated assumption that time-expanded graphs are very similar to road networks does not hold: all examined speed up techniques perform completely differently on our road networks than on our real-world time-expanded datasets. However, road networks seem to be a good alternative for condensed graphs and thus also for the time-dependent model. We expect that an approach working well in a (dynamic) time-dependent road network will also perform well on (dynamic) time-dependent railway networks.

Concerning speed up techniques in general, we gained further and interesting insights by our extensive experimental study. Hierarchical approaches seem to have problems with high-density networks, the chosen metric has a high impact on the achieved speed ups, edge-labels are somewhat superior to node-labels, and small diameters yield big speed ups for bidirectional search. As a consequence, the choice of which technique to use highly depends on the scenario. However, of all examined speed up techniques, ALT provides a reasonable trade-off between preprocessing time and space on the one hand and achieved speed up on the other hand. Although this approach is slower on hierarchical inputs it is more robust with respect to the input. In addition, ALT works in dynamic and time-dependent scenarios. Astonishingly, SHARC performs very well on most inputs, although it is a unidirectional technique. However, SHARC can easily be made bidirectional and hence, we assume that bidirectional SHARC would yield even faster queries on most inputs, including small-world graphs on which unidirectional SHARC fails.

Acknowledgments

The authors thank Dominik Schultes for providing the Highway Hierarchies code and his help on parameter settings. They also thank Robert Görke and Bastian Katz for providing data and Daniel Karch for implementing Arc-Flags. Finally, they thank Douglas Shier for many valuable suggestions of improvement.

REFERENCES

[1] Y. An, J. Janssen, and E.E. Milios, Characterizing and mining the citation graph of the computer science literature, Knowl Inform Syst 6 (2004), 664–678.

[2] D.A. Bader and K. Madduri, "Snap, small-world network analysis and partitioning: An open-source parallel graph framework for the exploration of large-scale networks," Proc. 22nd Int Parallel Distrib Process Symp (IPDPS'08), IEEE Computer Society, Miami, Florida, 2008, pp. 1–12.

[3] H. Bast, S. Funke, P. Sanders, and D. Schultes, Fast routing in road networks with transit nodes, Sci 316 (2007), 566–566.

[4] R. Bauer and D. Delling, SHARC: Fast and robust unidirectional routing, Proc 10th Workshop Algorithm Eng Experiments (ALENEX'08), SIAM, San Francisco, California, April 2008, pp. 13–26.

[5] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, Combining hierarchical and goal-directed speed up techniques for Dijkstra's algorithm, Proc 7th Workshop Experimental Algorithms (WEA'08), Vol. 5038 of Lecture Notes in Computer Science, Springer, Provincetown, Massachusetts, June 2008, pp. 303–318.

[6] R. Bauer, D. Delling, and D. Wagner, Experimental study on speed up techniques for timetable information systems, Proc 7th Workshop Algorithmic Approaches Transportation Modeling, Optim, Syst (ATMOS'07), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007, pp. 209–225.

[7] A. Berger, D. Delling, A. Gebhardt, and M. Müller–Hannemann, Accelerating time-dependent multi-criteria timetable information is harder than expected, Proc 9th Workshop Algorithmic Approaches Transportation Modeling, Optim, Syst (ATMOS'09), 2009, Dagstuhl Seminar Proceedings, Kopenhagen, Denmark.

[8] CAIDA: The Cooperative Association for Internet Data Analysis, Available at:. http://www.caida.org/, 2007.

[9] K. Cooke and E. Halsey, The shortest route through a network with time-dependent intermodal transit times, J Math Anal Appl 14 (1966), 493–498.

[10] J.R. Crobak, J.W. Berry, K. Madduri, and D.A. Bader, "Advanced shortest paths algorithms on a massively-multithreaded architecture," Workshop Multithreaded Architectures Appl (MTAAP 2007), Long Beach, California, 2007, pp. 1–8.

[11] G.B. Dantzig, Linear programming and extensions, Princeton University Press, Princeton, New Jersey, 1962.

[12] DBLP - database systems and logic programming. Available at: http://dblp.uni-trier.de/, 2007.

[13] D. Delling, "Time-dependent SHARC-routing," Proc 16th Ann Eur Symp Algorithms (ESA'08), Vol. 5193 of Lecture Notes in Computer Science, Springer, Karlsruhe, Germany, September 2008, pp. 332–343.

[14] D. Delling, K. Giannakopoulou, D. Wagner, and C. Zaroliagis, Timetable information updating in case of delays: Modeling issues, Technical Report 133, Arrival Technical Report, Patras, Greece, 2008.

[15] D. Delling, M. Holzer, K. Müller, F. Schulz, and D. Wagner, "High-performance multilevel routing, The shortest path problem: Ninth DIMACS implementation challenge," C. Demetrescu, A.V. Goldberg, and D.S. Johnson (Editors), American Mathematical Society, Rutgers, New Jersey, 2009, Vol. 74 of DIMACS Book, pp. 73–92.

[16] D. Delling and G. Nannicini, "Bidirectional core-based routing in dynamic time-dependent road networks," Proc 19th Int Symp Algorithms Computation (ISAAC'08), Vol. 5369 of Lecture Notes in Computer Science, Springer, Gold Coast, Australia, December 2008, pp. 813–824.

[17] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms, Algorithmics of large and complex networks," J. Lerner, D. Wagner, and K.A. Zweig (Editors), Springer, Heidelberg, Germany, 2009, pp. 117–139.

[18] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Highway hierarchies star, The shortest path problem: Ninth DIMACS implementation challenge," C. Demetrescu, A.V. Goldberg, and D.S. Johnson (Editors), American Mathematical Society, Rutgers, New Jersey, 2009, pp. 141–174.

[19] D. Delling and D. Wagner, "Landmark-based routing in dynamic graphs," Proc 6th Workshop Experimental Algorithms (WEA'07), Vol. 4525 of Lecture Notes in Computer Science, Springer, Rome, Italy, June 2007, pp. 52–65.

[20] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer Mathematik 1 (1959), 269–271.

[21] A.V. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," Proc 16th Ann ACM–SIAM Symp Discr Algorithms (SODA'05), Vancouver, Canada, 2005, pp. 156–165.

[22] A.V. Goldberg, H. Kaplan, and R.F. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," Proc 8th Workshop Algorithm Eng Exp (ALENEX'06), SIAM, Miami, Florida, 2006, pp. 129–143.

[23] A.V. Goldberg, H. Kaplan, and R.F. Werneck, "Better landmarks within reach," Proc 6th Workshop Exp Algorithms (WEA'07), Vol. 4525 of Lecture Notes in Computer Science, Springer, Rome, Italy, June 2007, pp. 38–51.

[24] A.V. Goldberg and R.F. Werneck, "Computing point-to-point shortest paths from external memory," Proc 7th Workshop Algorithm Eng Experiments (ALENEX'05), SIAM, Vancouver, Canada, 2005, pp. 26–40.

[25] R.J. Gutman, "Reach-based routing: A new approach to shortest path algorithms optimized for road networks," Proc 6th Workshop Algorithm Eng Experiments (ALENEX'04), SIAM, New Orleans, Louisana, 2004, pp. 100–111.

[26] HaCon - Ingenieurgesellschaft mbH, Available at:. http://www.hacon.de, 2001.

[27] P.E. Hart, N. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans Syst Sci Cybernetics 4 (1968), 100–107.

[28] M. Holzer, F. Schulz, and D. Wagner, "Engineering multilevel overlay graphs for shortest-path queries," Proc 8th Workshop Algorithm Eng Experiments (ALENEX'06), SIAM, Miami, Florida, 2006, pp. 156–170.

[29] M. Holzer, F. Schulz, D. Wagner, and T. Willhalm, Combining speed up techniques for shortest-path computations, ACM J Exp Algorithmics 10 (2006, Article 2.5).

[30] M. Holzer, F. Schulz, and T. Willhalm, "Combining speed up techniques for shortest-path computations," Proc 3rd Workshop Exp Algorithms (WEA'04), Vol. 3059 of Lecture Notes in Computer Science, Springer, Rio de Janeiro, Brazil, 2004, pp. 269–284.

[31] G. Karypis, Metis – family of multilevel partitioning algorithms. Available at: http://glaros.dtc.umn.edu/gkhome/views/metis.

[32] E. Köhler, R.H. Möhring, and H. Schilling, "Acceleration of shortest path and constrained shortest path computation," Proc 4th Workshop Experimental Algorithms (WEA'05), Vol. 3503 of Lecture Notes in Computer Science, Springer, Santorini Island, Greece, 2005, pp. 126–138.

[33] U. Lauther, "An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background," Geoinformation und mobilität - von der forschung zur praktischen anwendung, IfGI prints, 2004, Vol. 22, pp. 219–230.

[34] K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances," Proc 9th Workshop Algorithm Eng Experiments (ALENEX'07), SIAM, New Orleans, Louisana, 2007, pp. 23–35.

[35] K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, "Parallel shortest path algorithms for solving large-scale instances," The shortest path problem: Ninth DIMACS implementation challenge, C. Demetrescu, A.V. Goldberg, and D.S. Johnson (Editors), American Mathematical Society, Rutgers, New Jersey, 2009, Vol. 74 of DIMACS Book, pp. 249–290.

[36] R.H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm, "Partitioning graphs to speed up Dijkstra's algorithm," Proc 4th Workshop Exp Algorithms (WEA'05), Vol. 3503 of Lecture Notes in Computer Science, Springer, Santorini Island, Greece, 2005, pp. 189–202.

[37] R.H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm, Partitioning graphs to speedup Dijkstra's algorithm, ACM J Exp Algorithmics 11 (2006), Article 2.8.

[38] M. Müller–Hannemann, M. Schnee, and L. Frede, Efficient on-trip timetable information in the presence of delays, Proc 8th Workshop Algorithmic Approaches Transportation Modeling, Optim, Syst (ATMOS'08), Internationales Begegnungs-und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, September 2008, Dagstuhl Seminar Proceedings.

[39] M. Müller–Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, "Timetable information: Models and algorithms," Algorithmic methods for railway optimization, Springer, Heidelberg, Germany, 2007, Vol. 4359 of Lecture Notes in Computer Science, pp. 67–90.

[40] G. Nannicini, D. Delling, L. Liberti, and D. Schultes, "Bidirectional A* search for time-dependent fast paths," Proc 7th Workshop Exp Algorithms (WEA'08), Vol. 5038 of Lecture Notes in Computer Science, Springer, Provincetown, Massachusetts, June 2008, pp. 334–346.

[41] F. Pellegrini, Scotch: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package. Available at: http://www.labri.fr/perso/pelegrin/scotch/, 2007.

[42] PTV AG - Planung Transport Verkehr. Available at: http://www.ptv.de, 2005.

[43] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Towards realistic modeling of time-table information through the time-dependent approach," Proc 3rd Workshop Algorithmic MeThods Models Optim RailwayS (ATMOS'03), Budapest, Hungary, 2004, pp. 85–103.

[44] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, Efficient models for timetable information in public transportation systems, ACM J Exp Algorithmics 12 (2007), Article 2.4.

[45] P. Sanders and D. Schultes, "Engineering highway hierarchies," Proc 14th Ann Eur Symp Algorithms (ESA'06), Vol. 4168 of Lecture Notes in Computer Science, Springer, Zurich, Switzerland, 2006, pp. 804–816.

[46] D. Schultes and P. Sanders, "Dynamic highway-node routing," Proc 6th Workshop Experimental Algorithms (WEA'07), Vol. 4525 of Lecture Notes in Computer Science, Springer, Rome, Italy, June 2007, pp. 66–79.

[47] F. Schulz, D. Wagner, and K. Weihe, Dijkstra's algorithm online: An empirical case study from public railroad transport, ACM J Exp Algorithmics 5 (2000), Article 12.

[48] F. Schulz, D. Wagner, and C. Zaroliagis, "Using multi-level graphs for timetable information in railway systems," Proc 4th Workshop Algorithm Eng Exp (ALENEX'02), Vol. 2409 of Lecture Notes in Computer Science, Springer, San Francisco, California, 2002, pp. 43–59.

[49] D. Wagner, T. Willhalm, and C. Zaroliagis, Geometric containers for efficient shortest-path computation, ACM J Exp Algorithmics 10 (2005), Article 1.3.