Engineering Route Planning Algorithms[⋆]

Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
{delling,sanders,wagner}@ira.uka.de, mail@dominik-schultes.de

Abstract. Algorithms for route planning in transportation networks have recently undergone a rapid development, leading to methods that are up to three million times faster than Dijkstra's algorithm. We give an overview of the techniques enabling this development and point out frontiers of ongoing research on more challenging variants of the problem that include dynamically changing networks, time-dependent routing, and flexible objective functions.

1 Introduction

Computing an optimal route in a transportation network between specified source and target nodes is one of the showpieces of real-world applications of algorithmics. We frequently use this functionality when planning trips with cars or public transportation. There are also many applications like logistic planning or traffic simulation that need to solve a huge number of shortest-path queries in transportation networks. In the first part of this paper, we focus on the simplest case, a static *road* network with a fixed cost for each edge. The cost function may be any mix of travel time, distance, toll, energy consumption, scenic value, etc. associated with the edges. Some of the techniques described below work best if the cost function is positively correlated with travel time. The task is to compute the costs of optimal paths between arbitrary source-target pairs. Some preprocessing is allowed but it has to be sufficiently fast and space efficient to scale to the road network of a continent.

The main part of this paper is Section 2, which explains the ideas behind several practically successful speedup techniques for exact static routing in road networks. Section 3 makes an attempt to summarize the development of performance over time. In Section 4 we outline generalizations for public transportation, mobile devices, outputting optimal paths, and dynamically changing networks. Augmenting static techniques to time-dependent scenarios is discussed in Section 5, while Section 6 describes some experiences we made with implementing route planning algorithms for large networks. Then, Section 7 explains our experimental approach giving several examples by applying it to some algorithms we implemented. We conclude in Section 8 with a discussion of future challenges.

⋆ Partially supported by DFG grants SA 933/1-3, SA 933/5-1, WA 654/16-1 and the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL). A preliminary version of this paper has been published in [60].

2 Static Routing

We consider directed graphs $G = (V, E)$ with n nodes and $m = \Theta(n)$ edges. An edge (u, v) has the nonnegative edge weight $w(u, v)$. A shortest-path query between a source node s and a target node t asks for the minimum weight $d(s, t)$ of any path from s to t. In static routing, the edge weights do not change so that it makes sense to perform some *precomputations*, store their results, and use this information to accelerate the queries. Obviously, there is some tradeoff between query time, preprocessing time, and space for preprocessed information. In particular, for large road networks it would be prohibitive to precompute and store shortest paths between all pairs of nodes.

2.1 "Classical Results"

Dijkstra's Algorithm [20] – the classical algorithm for route planning – maintains an array of *tentative distances* $D[u] \geq d(s, u)$ for each node. The algorithm *visits* (or *settles*) the nodes of the road network in the order of their distance to the source node and maintains the invariant that $D[u] = d(s, u)$ for visited nodes. We call the rank of node u in this order its *Dijkstra rank* $\text{rk}_s(u)$. When a node u is visited, its outgoing edges (u, v) are *relaxed*, i.e., $D[v]$ is set to $\min(D[v], d(s, u)+w(u, v))$. Dijkstra's algorithm terminates when the target node is visited. The size of the search space, i.e. the number of settled nodes, is $O(n)$ and $n/2$ (nodes) on the average. We will assess the quality of route planning algorithms by looking at their *speedup* compared to Dijkstra's algorithm, i.e., how many times faster they can compute shortest-path distances.

Priority Queues. A naive implementation of Dijkstra's algorithm has a running time of $O(n^2)$ since finding the next node to settle takes $O(n)$ (linear search of candidates). However, the algorithm can be implemented using $O(n)$ priority queue operations. In the comparison based model this leads to $O(n \log n)$ execution time. In other models of computation (e.g. [71]) and on the average [47], better bounds exist. However, in practice the impact of priority queues on performance for large road networks is rather limited since cache faults for accessing the graph are usually the main bottleneck. In addition, our experiments indicate that the impact of priority queue implementations diminishes with advanced speedup techniques that dramatically reduce the queue sizes.

Bidirectional Search executes Dijkstra's algorithm simultaneously forward from the source and backwards from the target. Once some node has been visited from both directions, the shortest path can be derived from the information already gathered [12]. Bidirectional search can be combined with most other speedup techniques. On the other hand, it is a necessary ingredient of many advanced techniques.

Geometric Goal Directed Search (A^*). The intuition behind goal directed search is that shortest paths 'should' lead in the general direction of the target. A^* search [32] achieves this by modifying the weight of edge (u, v) to

$w(u, v) - \pi(u) + \pi(v)$ where $\pi(v)$ is a lower bound on $d(v, t)$. Note that this manipulation shortens edges that lead towards the target. Since the added and subtracted *vertex potentials* $\pi(v)$ cancel along any path, this modification of edge weights preserves shortest paths. Moreover, as long as all edge weights remain nonnegative, Dijkstra's algorithm can still be used. The classical way to use A^* for route planning in road maps estimates $d(v, t)$ based on the Euclidean distance between v and t and the average speed of the fastest road anywhere in the network. Since this is a very conservative estimation, the speedup for finding quickest routes is rather small. Goldberg et al. [25] even report a *slow-down* of more than a factor of two since the search space is not significantly reduced but a considerable overhead is added.

2.2 Exploiting Hierarchy

Small Separators. Transportation networks are almost planar, i.e., most edges intersect only at nodes. Hence, techniques developed for planar graphs will often also work for road networks. Using $O(n \log^2 n)$ space and preprocessing time, query time $O(\sqrt{n} \log n)$ can be achieved [22,41] for directed planar graphs without negative cycles. Queries accurate within a factor $(1 + \epsilon)$ can be answered in near constant time using $O((n \log n)/\epsilon)$ space and preprocessing time [70]. Most of these theoretical approaches look difficult to use in practice since they are complicated and need superlinear space. The approach from [70] has recently been implemented and experimentally evaluated on a road network with one million nodes [52]. While the query times are very good (less than $20\,\mu$s for $\epsilon = 0.01$), the preprocessing time and space consumption are quite high (2.5 hours and 2 GB, respectively).

Multi-Level Techniques. The first published practical approach to fast route planning [67,68] uses a set of nodes V_1 whose removal partitions the graph $G = G_0$ into small components. Now consider the *overlay graph* $G_1 = (V_1, E_1)$ where edges in E_1 are *shortcuts* corresponding to shortest paths in G that do not contain nodes from V_1 in their interior. Routing can now be restricted to G_1 and the components containing s and t respectively. This process can be iterated yielding a multi-level method [69,35,36,34]. A limitation of this approach is that the graphs at higher levels become much more dense than the input graphs thus limiting the benefits gained from the hierarchy. Also, computing small separators and shortcuts can become quite costly for large graphs.

Reach-Based Routing. Let $R(v) := \max_{s,t \in V} R_{st}(v)$ denote the *reach* of node v where $R_{st}(v) := \min(d(s, v), d(v, t))$. Gutman [31] observed that a shortest-path search can be stopped at nodes with a reach too small to get to source or target from there. Variants of reach-based routing work with the reach of edges or characterize reach in terms of geometric distance rather than shortest-path distance. The first implementation had disappointing speedups (e.g. compared to [67]) and preprocessing times that would be prohibitive for large networks.

Highway Hierarchies (HHs) [58,59] group nodes and edges in a hierarchy of levels by alternating between two procedures: Contraction (i.e., node reduction) removes low degree nodes by bypassing them with newly introduced shortcut edges. In particular, all nodes of degree one and two are removed by this process. Edge reduction removes *non-highway edges*, i.e., edges that only appear on shortest paths *close* to source or target. More specifically, every node v has a neighborhood radius $r(v)$ that we are free to choose. An edge (u, v) is a highway edge if it belongs to some shortest path P from a node s to a node t such that (u, v) is neither fully contained in the neighborhood of s nor in the neighborhood of t, i.e., $d(s, v) > r(s)$ and $d(u, t) > r(t)$. In all our experiments, neighborhood radii are chosen such that each neighborhood contains a certain number H of nodes. H is a tuning parameter that can be used to control the rate at which the network shrinks. The query algorithm is very similar to bidirectional Dijkstra search with the difference that certain edges need not be expanded when the search is sufficiently far from source or target. HHs were the first speedup technique that could handle the largest available road networks giving query times measured in milliseconds. There are two main reasons for this success: Under the above contraction routines, the road network remains sparse and near planar. Furthermore, preprocessing can be done using limited local searches starting from each node which resulted in the fastest preprocessing at that time.

Advanced Reach-Based Routing. It turns out that the preprocessing techniques developed for HHs can be adapted to preprocessing reach information [26]. This makes reach computation faster and more accurate. More importantly, shortcuts make queries more effective by reducing the number of nodes traversed and by reducing the reach-values of the nodes bypassed by shortcuts. Reach-based routing is slower than HHs both with respect to preprocessing time and query time. However, the latter can be improved by a combination with goal-directed search to a point where both methods have similar performance.

Highway-Node Routing (HNR). In [65] the multi-level routing scheme with overlay graphs [67,69,35,36] is generalized so that it works with arbitrary sets of nodes rather than only with separators. This is achieved using a new query algorithm that stalls suboptimal branches of search on lower levels of the hierarchy. By using only *important* nodes for higher levels, query performance is comparable to HHs. Preprocessing is done in two phases. In the first phase, nodes are classified into levels. In the second phase, the shortcuts are recursively computed bottom up. Shortcuts from level ℓ are found by local searches in level $\ell - 1$ starting from nodes in level ℓ. This second phase is very fast and easy to update when edge weights change.

Contraction Hierarchies (CHs) are a special case of highway-node routing where we have n levels – one level for each node [23]. Such a fine hierarchy can improve query performance by a considerable factor. The queries are also further simplified since it is sufficient to search upward in a *search graph.* This property saves considerable space since each edge is only stored at its lower endpoint.

CH preprocessing proceeds in two phases. The first phase orders nodes by importance. The second phase contracts (removes) nodes in this order. When node v is contracted it is removed from the network in such a way that shortest path distances between the remaining nodes are preserved. See Fig. 1 for an example. Local searches are used to decide which of the potential shortcuts of the form $\langle u, v, w \rangle$ are needed. In a sense, contraction hierarchies are a simplification of HHs where only the node contraction phases are used (using a more careful implementation). Node ordering keeps the nodes in a priority queue with priorities based on how attractive it is to contract a node. The most important term of the priority function is the *edge difference* – how many additional edges would the graph get if a node is contracted (this value may be negative). Another important priority term ensures that nodes are removed from the networks *uniformly* rather than only in one area of the network. Due to their simplicity and efficiency, contraction hierarchies are now used in almost all of our advanced routing techniques.

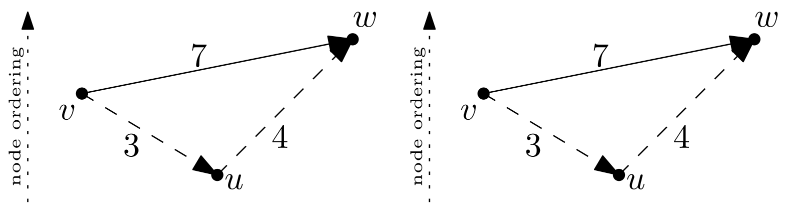

Fig. 1. Contraction with node ordering $u < v < w$. Node u is contracted by adding a shortcut from v to w and by removing the incoming and outgoing edges of u.

Distance Tables. Once a hierarchical routing technique (e.g., HH, HNR, CH) has shrunk the size of the remaining network G' to $\Theta(\sqrt{n})$, one can afford to precompute and store a complete distance table for the remaining nodes [59]. Using this table, one can stop a query when it has reached G'. To compute the shortest-path distance, it then suffices to lookup all shortest-path distances between nodes entering G' in forward and backward search respectively. Since the number of entrance nodes is not very large, one can achieve a speedup close to two compared to the underlying hierarchical technique.

Transit-Node Routing precomputes not only a distance table for important (*transit*) nodes but also all relevant connections between the remaining nodes and the transit nodes. Independently, three approaches proved successful for selecting transit nodes: separators [51,15], border nodes of a partition [3,4,2], and nodes categorized as important by other speedup techniques [3,4,61]. It turns out that for route planning in road networks, the latter approach is the most promising one. Since only about 7–10 such *access connections* are needed per node one can 'almost' reduce routing in large road networks to about 100 table lookups. Interestingly, the difficult queries are now the local ones where the shortest path does not touch any transit node. This problem can be solved by introducing several *layers* of transit nodes. Between lower layer transit nodes, only those routes need to be stored that do not touch the higher layers. Transit-node routing (e.g., using appropriate slices of a CH) reduces routing times to a few microseconds at the price of larger preprocessing time and additional space consumption.

2.3 Advanced Goal-Directed Search

Edge Labels. The idea behind edge labels is to precompute information for an edge e that specifies a set of nodes $M(e)$ with the property that $M(e)$ is a superset of all nodes that lie on a shortest path starting with e. In an s–t query, an edge e need not be relaxed if $t \notin M(e)$. In [67], $M(e)$ is specified by an *angular range*. More effective is information that can distinguish between long range and short range edges. In [74] many *geometric containers* are evaluated. Very good performance is observed for axis parallel rectangles. A disadvantage of geometric containers is that they require a complete all-pairs shortest-path computation. Faster precomputation is possible by partitioning the graph into k regions that have similar size and only a small number of boundary nodes. Now $M(e)$ is represented as a k-vector of *edge flags* [44,43,48,49,63] where flag i indicates whether there is a shortest path containing e that leads to a node in region i. Fig. 2 gives an example. Edge flags can be computed using a single-source shortest-path computation from all boundary nodes of the regions. A further improvement gets by with only one (though comparatively expensive) search for each region [33].

SHARC is an extension of the edge flag approach [6]. By using subroutines from hierarchical approaches—namely contraction—during preprocessing most of the disadvantages of edge flags can be remedied. The key observation is that it is sufficient to set suboptimal edge flags to many edges of the graph. Expensive preprocessing is then only done for important edges. In addition, SHARC extends the idea of the 2-level edge flag approach presented in [48] to a multi-level arc-flags setup. The result is a fast *unidirectional* query algorithm, which is especially advantageous in scenarios where bidirectional search is prohibitive, e.g. time-dependent networks (cf. Section 5). In general, SHARC can be interpreted as

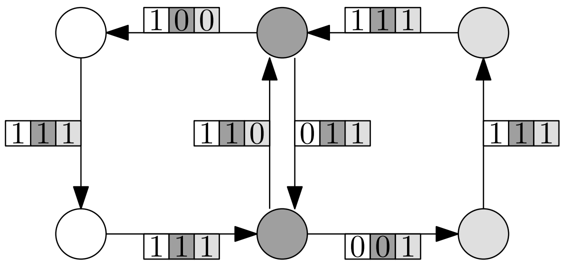

Fig. 2. Example for Arc-Flags. The graph is partitioned in 3 regions.

goal-directed technique that incorporates hierarchical aspects implicitely. Even faster query times can be achieved by a bidirectional version of SHARC.

Landmark A^* (ALT). Using the triangle inequality, quite strong bounds on shortest-path distances can be obtained by precomputing distances to a set of *landmark* nodes (≈ 20) that are well distributed over the far ends of the network [25,29]. Using reasonable space and much less preprocessing time than for edge labels, these lower bounds yield considerable speedup for route planning.

Precomputed Cluster Distances (PCD). In [46], a different way to use precomputed distances for goal-directed search is given. The network is partitioned into clusters and then a shortest connection between any pair of clusters U and V, i.e., $\min_{u \in U, v \in V} d(u, v)$, is precomputed. PCDs cannot be used together with A^* search since reduced edge weights can become negative. However, PCDs yield upper and lower bounds for distances that can be used to prune search. This gives speedup comparable to landmark-A^* using less space. Using the many-to-many routing techniques outlined in Section 4, cluster distances can also be computed efficiently.

2.4 Combinations

Bidirectional search can be profitably combined with almost all other speedup techniques. Indeed, it is a required[1] ingredient of highway hierarchies, transit-node routing, highway-node routing and contraction hierarchies and it achieves a considerable improvement for reach-based routing and edge flags. Willhalm et al. have made a systematic comparison of combinations of pre-2004 techniques [38,76,37]. Landmark A^* harmonizes very well with reach-based routing [26] whereas it gives only a small additional speedup when combined with HHs [17].

Recently, the combination of edge flags with hierarchical approaches has proved very successful [8]. Contraction hierarchies combined with edge flags yield query times almost as low as transit-node routing using less space and preprocessing time. This large reduction in preprocessing times compared to edge-flags alone stems from a restriction of the edge-flag computations to a network that is already considerably contracted. This combination is also quite effective for many types of networks where hierarchical methods alone work not as well as for road networks. Edge flags combined with transit-node routing lead to the currently fastest query times for road networks.

2.5 Differences to Commercial Systems

It turns out that commercial route planning systems and the methods described above have a lot of things in common but differ in a crucial fact. Like commercial systems, the methods from Sections 2.2 and 2.3 follow some kind of intuition a

[1] However, there are approaches that combine forward search with a type of backward graph exploration that is not shortest path search. This will become relevant in Section 5.

human uses for traveling in transportation networks. The main difference is that commercial systems might provide suboptimal results while the latter guarantee to find the shortest path. Surprisingly, settling for approximate results does not result in faster query times. In fact, the contrary is true.

For example, a common approach in commercial car navigation systems is the following: do not look at 'unimportant' streets, unless you are close to the source or target [39]. This heuristic needs careful hand tuning of road classifications to produce reasonable results but yields considerable speedups. Recalling the concept of highway hierarchies, one might notice that HH follows the same intuition: after a fixed number of hops, consider only a highway network. It turns out that one main reason for the better performance of HH compared to the heuristic is the correctness of the former. The latter has to make a precarious compromise between quality and size of the search space that relies on manual classification of the edges into levels of the hierarchy. In contrast, after setting a few quite robust tuning parameters, HH-preprocessing automatically computes a hierarchy aggressively tuned for high performance.

3 Chronological Summary – The Horse Race

Although academic research on speedup techniques for DIJKSTRA's algorithm started in the late nineties, motivated from timetable information [67], a boost in development was the publication of continental-sized road networks in 2005. The European network was made available for scientific use by the company PTV AG. The USA network was from publicly available geographical data [72]. Before that, input-sizes were limited and even worse, most data was confidential, e.g. the timetable data used in [67]. This lead to the problem that it was hard to compare different approaches with respect to performance. Once large road networks were available to everybody [58,19] so that speedup techniques became comparable, a "horse race" started: Which group can achieve the fastest query (and preprocessing) times on these inputs. In this Section we give a chronological summary of this race, including techniques that were published *before* 2005.

However, it is still difficult to compare speedup techniques even for road networks because there is a complex tradeoff between query time, preprocessing time and space consumption that depends on the network, on the objective function, and on the distribution of queries. Still, we believe that some ranking helps to compare the techniques. We take the liberty to speculate on the performance of some older methods that have never been run on such large graphs and whose actual implementations might fail when one would attempt it. In Tab. 1 we list speedup techniques in chronological order that are 'best' with respect to speedup for random queries and the largest networks tackled at that point. Sometimes we list variants with slower query times if they are considerably better with respect to space consumption or manageable graph size.

Before [67] the best method would have been a combination of bidirectional search with geometric A^* yielding speedups of 2–3 over unidirectional Dijkstra. The separator-based multi-level method from [67] can be expected to work

Table 1. Chronological development of the fastest speedup techniques. As date for the first publication, we usually give the submission deadline of the respective conference. If available, we always selected measurements for the European road network even if they were conducted after the first publication. Otherwise, we *linearly* extrapolated the preprocessing times to the size of Europe, which can be seen as a *lower bound*. Note that not all speedup techniques have been preprocessed on the same machine. Also note that we report the space consumption of the technique *including* the graph.

method	first pub.	date mm/yy	data from	size $n/10^6$	space [B/n]	preproc. [min]	speedup
DIJKSTRA	[20]	08/59	-	18	21	0	1
separator multi-level	[67]	04/99	[36]	0.1	?	$> 5\,400$	52
edge flags (basic)	[44]	03/04	[45]	1	35	299	523
landmark A^*	[25]	07/04	[28]	18	89	13	28
edge flags	[43,48]	01/05	[33]	18	30	1 028	3 951
HHs (basic)	[58]	04/05	[58]	18	49	161	2 645
reach + shortc. + A^*	[26]	10/05	[28]	18	100	1 625	1 559
	[28]	08/06	[28]	18	56	141	3 932
HHs + dist. tab.	[59]	04/06	[64]	18	68	13	10 364
HHs + dist. tab. + A^*	[17]	08/06	[64]	18	92	14	12 902
high-perf. multi-level	[51]	06/06	[15]	18	181	1 440	401 109
transit nodes (eco)	[3]	10/06	[64]	18	140	25	574 727
transit nodes (gen)	[3]	10/06	[64]	18	267	75	1 470 231
highway nodes	[65]	01/07	[64]	18	28	15	7 437
approx. planar $\epsilon = 0.01$	[52]	09/07	[52]	1	2 000	150	18 057
SHARC	[6]	09/07	[7]	18	34	107	21 800
bidirectional SHARC	[6]	09/07	[7]	18	41	212	97 261
contr. hier. (aggr)	[23]	01/08	[23]	18	17	32	41 051
contr. hier. (eco)	[23]	01/08	[23]	18	21	10	28 350
CH + edge flags (aggr)	[8]	01/08	[8]	18	32	99	371 882
CH + edge flags (eco)	[8]	01/08	[8]	18	20	32	143 682
transit nodes + edge flags	[8]	01/08	[8]	18	341	229	3 327 372
contr. hier. (mobile)	[62]	04/08	[62]	18	8	31	9 878

even for large graphs if implemented carefully. Computing geometric containers [67,74,75] is still infeasible for large networks. Otherwise, they would achieve much larger speedups than the separator-based multi-level method. Until recently, computing edge flags has also been too expensive for Europe and the USA but speedups beyond 523 have been observed for a graph with one million nodes [49]. Landmark A^* works well for large graphs and achieves average speedup of 28 using reasonable space and preprocessing time [25]. The implementation of HHs [58] was the first that was able to handle Europe and the USA. This implementation wins over all previous methods in almost all aspects. A combination of reach-based routing with landmark A^* [26] achieved better query times for the USA at the price of a considerably higher preprocessing time. At first, that code did not work well on the European network because it is difficult to handle the present long-distance ferry connections, but later it could be considerably improved [27]. By introducing distance tables and numerous other improvements, highway hierarchies took back the lead in query time [59] at the same time using an order of magnitude less preprocessing time than [58]. The cycle of innovation accelerated even further in 2006. Müller [51] aggressively precomputes the pieces of the search space needed for separator-based multi-level routing. At massive expense of space and preprocessing time, this method can achieve speedups around 400 000. (The original implementation cannot directly measure this because it has large overheads for disk access and parsing of XML-data). Independently, transit-node routing was developed [3], that lifts the speedup to six orders of magnitude and completely replaces Dijkstra-like search by table lookups. This approach was further accelerated by combining transit nodes with edge flags [8], yielding speedups of over 3 millions.

However, the story was not over because speedup techniques have to fulfill several properties, in addition to speed, if they should be used in real-world scenarios: Memory consumption should be as low as possible, simple updating of preprocessing due to traffic jams should be possible, and speedup techniques should work in time-dependent networks. Highway-Node Routing [65] and contraction hierarchies [23,62] fulfill the first two requirements, while SHARC [6,14] was developed for the latter scenario. These scenarios are the topic of Sections 4 and 5.

4 Generalizations

Many-to-Many Routing. In several applications we need complete distance tables between specified sets of source nodes S and target nodes T. For example, in logistics optimization, traffic simulation, and also within preprocessing techniques [46,3]. Many non-goal-directed bidirectional search methods [67,26,65]) can be adapted in such a way that only a single forward search from each source node and a single backward search from each target node is needed [42]. The basic idea is quite simple: Store the backward search spaces. Arrange them so that each node v stores an array of pairs of the form $(t, d(v,t))$ for all target nodes that have v in their backward search space. When a forward search from s settles a node v, these pairs are scanned and used to update the tentative distance from s to t. This is very efficient because the intersection between any two forward and backward search spaces is small and because scanning an array is much faster than priority queue operations and edge relaxations governing the cost of Dijkstra's algorithm. For example, for $|S| = |T| = 10\,000$, the implementation in [23] needs only about 10s.

Outputting Paths. The efficiency of many speedup techniques stems from introducing shortcut edges and distance table entries that replace entire paths in the original graph [67,58,26,59,51,3]. A disadvantage of these approaches is that the search will output only a 'summary description' of the optimal path that involves shortcuts. Fortunately, it is quite straightforward to augment the shortcuts with information for unpacking them [17,42,3,23,62]. Since one can afford to precompute unpacked representations of the most frequently needed long-distance shortcuts, outputting the path turns out to be up to four times *faster* than just traversing the edges in the original graph.

Mobile Route Planning. Most of the techniques described above can be adapted to mobile devices such as car navigation systems or cell phones. However, space is at a premium here and most of the data has to reside on an external memory device such as flash memory that has high access latencies and where write accesses are very expensive. Therefore, the first measure is to keep the priority queue and the nontrivial tentative distances in the fast memory. Goldberg and Werneck [29] successfully implemented the ALT algorithm on a Pocket PC. Their largest road network (North America, 29 883 886 nodes) occupies 3 735 MB and a random query takes 329 s. The RE algorithm [31,27] has been implemented on a mobile device, yielding query times of "a few seconds including path computation and search animation" and requiring "2–3 GB for USA/Europe" [24]. Contraction hierarchies have been implemented using careful blocking of nodes that are often accessed together and using aggressive variable-bitlength encoding for edges and edge weights [62]. This implementation needs only 140 MByte space for the European network and 69 ms query time on a 330 MHz ARM 11 processor without any preloaded information in the fast memory.

Turn Penalties. On most road crossings, going straight takes less time than, e.g., turning left. Such turn penalties (and disallowed turns) can be modelled using *edge based routing* where road segments become nodes of the graph and edges connect possible pairs of successive road segments. The disadvantage of this model is that a naive implementation takes several times more space than node-based routing. This problem can be circumvented by running logical edge-based routing based on bidirectional Dijkstra on a node-based physical representation. A thin interface layer makes an on-the-fly conversion. For a small subset of important nodes in the edge-based graph, an explicit contraction hierarchy is computed. The implicit Dijkstra search can switch to a fast contraction-hierarchy query when the search is covered by important nodes. Thus, most advantages of both approaches are combined [50,73].

Flexible Objective Functions. The objective function in road networks depends in a complex way on the vehicle (fast, slow, too heavy for certain bridges, etc.) the behavior and goals of the driver (cost sensitive, thinks he is fast, etc.), the load, and many other aspects. While the appropriate edge weights can be computed from a few basic parameters, it is not feasible to perform preprocessing for all conceivable combinations. Currently, our best answer to this problem is highway-node routing/contraction hierarchies [65,23]. Assuming that the important nodes are important for any reasonable objective function, only the second phase of preprocessing needs to be repeated. This is an order of magnitude faster than computing a HH or the node ordering of contraction hierarchies.

Dynamization. In online car navigation, we want to take traffic jams etc. into account. At first glance, this is fatal for most speedup techniques since even a single traffic jam can invalidate any of the precomputed information. However, we can try to selectively update only the information affected by the traffic jam and/or relevant to the queries at hand. Updating the preprocessing of Geometric Containers has been analyzed in [75]. By only allowing increases in the sizes of containers, queries stay correct, but performance may be worse than recomputing the containers from scratch. Landmark A^* can be dynamized either by noticing that lower bounds remain valid when edge weights can only increase, or by using known dynamic graph algorithms for updating the shortest-path trees from the landmarks [18]. Highway-node routing was originally developed for dynamization [65]. Indeed, it allows fast and easy updates (2–40 ms per changed edge weight depending on the importance of the edge). Even faster dynamization is possible by leaving the preprocessed information unchanged and only causing the query algorithm to descend the hierarchy when it encounters unreliable information. This approach scales to quite large numbers of *delays* using an iterative approach that only takes edges into account that actually affect the current query [64]. In the meantime this approach has been successfully adapted to mobile contraction hierarchies, generalizing [62].

Multi-Criteria Routing. The fastest route in transportation networks is often not the "best" one. For example, users traveling by train may be willing to accept longer travel times if the number of required transfers is lower or the cost of a journey with longer duration is cheaper. We end up in a multi-criteria scenario [53,57,21] in which none of the high-performance approaches developed in the last years can be applied easily. The adaption of a fast method to this scenario is one of the main challenges in the near future.

Multimodal Routing. Often, we use more than one transportation network for traveling. For example, we first use our car to get to the train station, use the train to get to the airport and finally board a plane. Planning a route in such a combined network has to fulfill certain constraints, e.g., your own car is only available at the beginning of the journey or we want to avoid switching the type of transportation too frequentely. These constraints can be modeled by adding appropriate labels to each edge [1,34]. Unfortunately, using a fast routing algorithm in such a *label-constrained* scenario is very complicated and hence, another challenging task.

5 Time-Dependency

As already mentioned, most developed techniques require the network to be *static* or only allow a small number of updates [65,18]. In practice, however, travel duration often depends on the departure time. It turns out that efficient models for routing in almost all transportation systems, e.g., timetable information for railways or scheduling for airplanes, are based on *time-dependent* networks. Moreover, road networks are not static either: there is a growing body of data on travel times of important road segments stemming from road-side sensors, GPS systems inside cars, traffic simulations, etc. Using this data, we can assign *speed profiles* to roads. This yields a time-dependent road network.

Switching from a static to a time-dependent scenario is more challenging than one might expect: The input size increases drastically as travel times on congested motorways change during the day. On the technical side, most static techniques rely on bidirectional search, i.e., a second search is started from the target. This concept is prohibited in time-dependent scenarios as the arrival time would have to be known in advance for such a procedure. Moreover, possible problem statements for shortest paths become even more complex in such networks. A user could ask at what time she should depart in order to spend as little time traveling as possible. As a result, none of the existing high-performance techniques can be adapted to this realistic scenario easily.

5.1 Modeling Issues

The major difference between static and time-dependent routing is the usage of functions instead of constants for specifying edge weights. We use piece-wise linear functions for modeling time-dependency in road networks[2]. Each edge gets assigned a number of sample points that depict the travel time on this road at the specific time. Evaluating the travel time for an edge at time τ is then done by linear interpolation between the points left and right to τ. In order to allow a polynomial time exact solution [40,56], we assume that the network fulfills the *FIFO property* or *non-overtaking property*: if A leaves an arbitrary node s before B, B cannot arrive at any node t before A.

Currently, research concentrates on solving the earliest arrival problem in time-dependent networks, i.e., find the quickest route from s to t for given time of departure τ. Note that this problem is closely related to the latest departure problem, i.e., find the quickest route from s to t such that you arrive at t at a given time τ' [13].

Rolling-Out Time-Dependency. Another approach to model time-dependency is to roll out the time component. In the *time-expanded* approach for timetable information [66,54], each time-dependent edge is multiplied such that each edge represents exactly one connection and a timestamp is assigned to each node. Although the resulting graphs are time-independent, the main downside of this approach is the increase in input size. While this is still practicable for timetable information [53,30], the time-expanded graphs get way too big for road networks. In addition, adaption of speedup techniques to the time-expanded approach is more complicated than expected [9]. Hence, the time-dependent approach seems more promising.

5.2 Basic Algorithmic Toolbox

Analyzing all speedup techniques, preprocessing relies on two major ingredients: (local) Dijkstra-searches and contraction. Hence, for correct routing in time-dependent networks, both ingredients need to be augmented.

Generalizing Dijkstra's Algorithm. A straightforward extension [11] of Dijkstra's algorithm is capable of computing the distance between s and t

[2] Note that this is flexible enough to accurately integrate time-tabled connections such as ferries.

when departing from s at time τ. However, we may also need to compute a *profile*, i.e. the distance between s and t for *all* departure times. It turns out that this is more expensive but can be computed by a label-correcting variant of DIJKSTRA's algorithm [13]: distance labels now become functions of time and edge relaxations compute minima of two time-dependent functions; nodes may have to be scanned several times when parts of the their distance label improve. An interesting result from [13] is the fact that the runtime of label-correcting algorithms highly depends on the complexity of the edge-functions.

Contraction. The second ingredient of high-performance speedup techniques is contraction (cf. Section 2). Basically, we may leave the principle untouched: unimportant nodes are removed from the graph and shortcuts are added in order to preserve distances between remaining nodes. However, contraction in time-dependent road networks is more expensive, in terms of space consumption, than in time-independent networks. Let $P(f)$ be the number of interpolation points of the function f assigned to an edge (u, v). Then the composed function $f \oplus g$, modeling the duration for traversing g after f, may have up to $P(f)+P(g)$ interpolation points in the worst case. This is one of the main problems when routing in time-dependent graphs: Almost all speedup techniques developed for static scenarios rely on adding long shortcuts to the graph. While this is "cheap" for static scenarios, the insertion of time-dependent shortcuts yields a high amount of preprocessed data.

An interesting observation is that in timetable networks, the problem of increasing interpolation does not exist. More precisely, $P(f \oplus g) = \min\{P(f), P(g)\}$ holds as the number of relevant departure times is dominated by the edge with less connections.

5.3 Adapting Speedup Techniques

Up to now, three different approaches have successfully been adapted to the time-dependent scenario. They either use a unidirectional search or perform a backward search that limits the node set the forward search has to visit.

A* with landmarks (ALT) was the first natural choice for adaption to time-dependent scenarios. Like for the dynamic time-independent scenarios, ALT performs correct queries as long as potentials are feasible. So, by using the lower bound of each edge during preprocessing, ALT-queries stay correct. Unfortunately, a unidirectional variant [18] only leads to a mild speedups of 3–5. A speedup of 30 can be obtained if we are willing to accept suboptimal paths being up to 15% longer than the shortest [55]. This can be achieved by a bidirectional approach. Here, a time-independent backward search bounds the node set that has to be examined by the forward search. This approach can be enhanced by performing ALT only on a small core built by contraction during preprocessing [16].

SHARC. A disadvantage of ALT is its rather small speedup even in static scenarios. This was the main motivation for the development of SHARC, a unidirectional speedup technique [6] being as fast as bidirectional approaches. It is based on two foundations: contraction and edge flag computation. An edge flag is now set as soon as it is important for at least one departure time [14]. As a result, quickest paths for all departure times between two points have arc-flags set to true. A straight-forward approach is to compute flags using the label correcting algorithm described above. While this works for large timetable graphs, preprocessing takes way too long for road networks. By setting suboptimal arc-flags, preprocessing times can be reduced but for the price of query performance. Depending on the degree of perturbation, time-dependent SHARC is up to 214 times faster than Dijkstra.

Contraction Hierarchies [5]. Due to their simplicity and good performance, contraction hierarchies are an interesting candidate for generalization. In the most simple implementation, the first preprocessing phase (node ordering) can be done on the static graph. Only the second phase needs to replace ordinary Dijkstra searches by profile searches. This is straight-forward in principle but challenging because a naive implementation is very expensive. The query algorithm is slightly less obvious. The simultaneous forward and backward Dijkstra-searches of the static version are replaced by four activities[3]: 1. A time dependent forward-upward Dijkstra-search. 2. Backward *exploration* of the downward edges leading to the target. Note that in most other hierarchical search techniques this weakening of the backward search would imply traversing the entire graph whereas the DAG (directed acyclic graph) property of contraction hierarchies makes it possible that only a small subset of the nodes is visited. 3. Pruning of supoptimal paths (when forward distance plus a lower bound on the distance to the target exceed some upper bound). 4. A forward-downward Dijkstra search starting simultaneously from all the promising places where forward search and backward exploration met that is confined to the edges explored in component 2. Various forms of upper and lower bounds on travel time can be used to improve pruning, accelerate preprocessing, and save space.

6 Implementation

Advanced algorithms for routing in road networks require thousands of lines of well written code and hence require considerable programming skill. In particular, it is not trivial to make the codes work for large networks. Here is an incomplete list of problems and complications that we have seen in routing projects: Graphs have to be glued together from several files. Tools for reading files crash for large graphs. Algorithm library code cannot handle large graphs at all. The code slows down several times when switching from a custom graph representation to an algorithm library. 32-bit code will not work. Libraries do not work with 64-bit code. Our conclusion from these experiences was to design our own graph data structures adapted to the problem at hand. We use C++

[3] We avoid the term ‘phase’ here since the activities can be interspersed in various ways.

with encapsulated abstract data types. Templates and inline functions make this possible without performance penalties.

Speedup techniques developed by algorithmicists usually come with high level arguments why they should yield optimal paths. While this is already much more robust than purely heuristic algorithms, we sometimes observed that subtle details only revealed by a detailed correctness proof can yield suboptimal paths. For example, we had several cases where the algorithm considered was only correct when all shortest paths are unique.

There are plenty of things that can go wrong both with the algorithms and their implementations. The implementation can help here with extensive consistency checks in assertions and experiments that are always checked against naive implementations, i.e., queries are checked against Dijkstra's algorithm and fast preprocessing algorithms are checked against naive or old implementations. On the long run one also needs a flexible visualization tool that can draw pieces of large graphs, paths, search spaces, and node sets. Since we could not find tools for this purpose that scale to large road networks, we implemented our own system [10].

7 Methodology of Experiments

Evaluating speedup techniques on large networks is not as trivial as one might expect. For example, an obvious choice of queries uses randomly selected node pairs on the largest available graph. Although the resulting average query time is a meaningful number, it is not quite satisfactory since most queries will produce very long paths (thousands of kilometers) that are actually rare in practice. One possible solution is to use random queries on a variety of subgraphs. However, this leads to a plethora of arbitrary choices that make it difficult to compare results. In particular, authors will be tempted to choose only those subgraphs for which their method performs well.

Sets of real world queries would certainly be interesting, but it is unlikely that a sample taken from one server is actually representative for the entire spectrum of route planning applications. We therefore chose a more systematic approach [58] that has also been adopted in several other studies: We generate a random query with a specified 'locality' r by choosing a random starting node s, and a target node t with Dijkstra rank $\mathrm{rk}_s(t) = r$ (i.e., the r-th node visited by a Dijkstra search from s). In our studies, we generate many such queries for each r which is a power of two. We then plot the distribution with median, quartiles, and outliers for each of these values of r. For the European road network, Fig. 3 shows the results for contraction hierarchies, CHs combined with edge flags, and for transit-node routing. We view it as quite important to give information on the entire distribution since some speedup techniques have large fluctuations in query time.

In some cases, e.g., for contraction hierarchies, it is also possible to compute good upper bounds on the search space size of *all* queries that can ever happen for a given graph [59,64,23]. Figure 4 gives the upper bounds for contraction

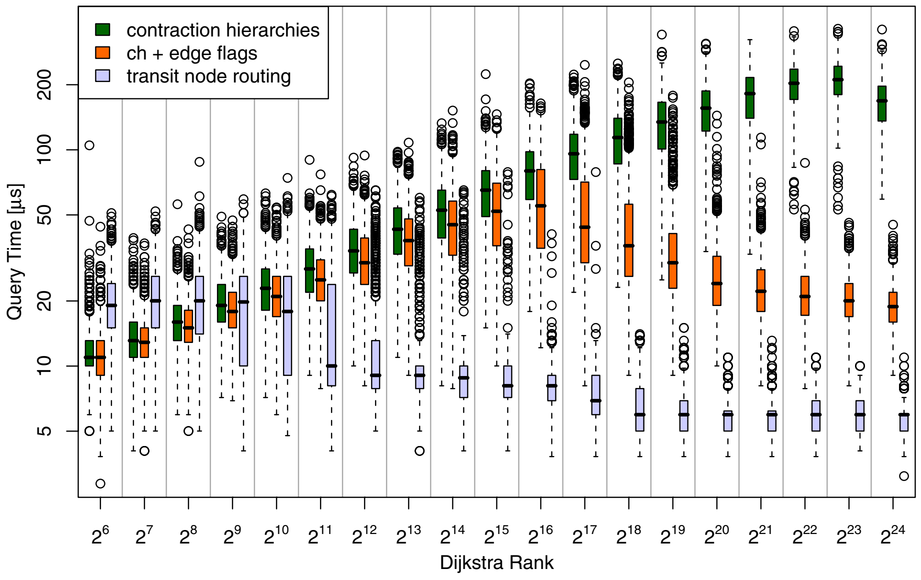

Fig. 3. Query performance of various speedup techniques against Dijkstra rank. We observe that adding edge flags to contraction hierarchies accelerates long-range queries, while transit node routing also accelerates mid-range queries (compared to pure contraction hierarchies).

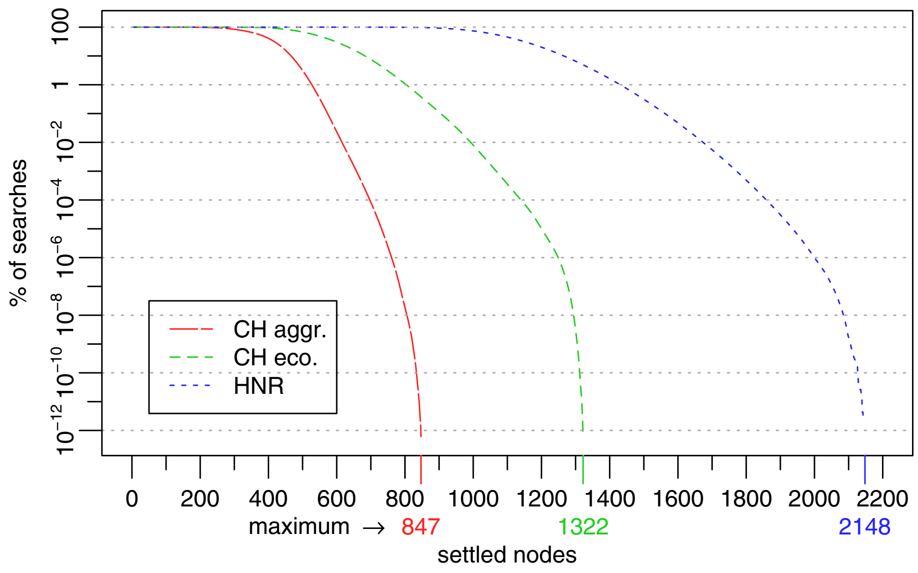

Fig. 4. Worst case upper bounds for various hierarchical speedup techniques

hierarchies and highway-node routing. We view this as a quite good surrogate for the absence of meaningful worst case upper bounds that would apply to all conceivable networks.

8 Conclusions and Open Problems

Speedup techniques for routing in static road networks have made tremendous progress in the last few years. While for challenging applications such as logistics planning and traffic simulation, query times cannot be fast enough, for other applications the query times are more than satisfying: other overheads like displaying routes or transmitting them over the network are the bottleneck once the query time is below a few milliseconds.

A major challenge is to close the gap to theory, e.g., by giving meaningful characterizations of 'well-behaved' networks that allow provably good worst-case bounds. In particular, we would like to know for which networks the existing techniques will also work, e.g., for communication networks, VLSI design, social networks, computer games, graphs derived from geometric routing problems, ...

Perhaps the main academic challenge is to go beyond static point-to-point routing. Although first techniques already provide promising results, the gap between static and time-dependent routing is still very big. A very important topic for the furture is to reduce preprocessing space of time-dependent techniques. Here, the main problem lies in the mentioned problem that shortcuts are "cheap" in static scenarios, while in time-dependent scenarios shortcuts can become highly complex objects. Further beyond that, we want multi-criteria optimization for individual paths and we want to run realistic traffic simulations.

Acknowledgements

We would like to thank our coauthors on route planning, Holger Bast, G. Veit Batz, Reinhard Bauer, Chris Barrett, Keith Bisset, Stefan Funke, Robert Geisberger, Martin Holzer, Ekkehard Köhler, Goran Konjevod, Sebastian Knopp, Leo Liberti, Madhav V. Marathe, Domagoj Matijevic, Jens Maue, Rolf Möhring, Kirill Müller, Matthias Müller-Hannemann, Giacomo Nannicini, Evangelia Pyrga, Dennis Schieferdecker, Heiko Schilling, Frank Schulz, Christian Vetter, Lars Volker, Karten Weihe, Thomas Willhalm, and Christos Zaroliagis for their valuable contributions. We also had many interesting discussions with Rob van den Berg, Sebastian Egner, Andrew Goldberg, Joaquim Gromicho, Stefan Hug, Ali Nowbakht Irani, Riko Jacob, Ulrich Lauther, Ramon Lentink, Dennis Luxen, Paul Perdon, Martin Pfeifle, Mikkel Thorup, Jaques Verrier, Peter Vortisch, and Renato Werneck. Finally, we thank PTV AG and HaCon for providing us with real-world data for scientific use.

References

1. Barrett, C., Bisset, K., Holzer, M., Konjevod, G., Marathe, M.V., Wagner, D.: Engineering Label-Constrained Shortest-Path Algorithms. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 27–37. Springer, Heidelberg (2008)
2. Bast, H., Funke, S., Matijevic, D.: TRANSIT Ultrafast Shortest-Path Queries with Linear-Time Preprocessing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
3. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Shortest-Path Queries in Road Networks. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 46–59. SIAM, Philadelphia (2007)
4. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast Routing in Road Networks with Transit Nodes. Science 316(5824), 566 (2007)
5. Batz, G.V., Geisberger, R., Sanders, P.: Time Dependent Contraction Hierarchies - Basic Algorithmic Ideas. Technical report, ITI Sanders, Faculty of Informatics, Universität Karlsruhe (TH) (2008), arXiv:0804.3947v1 [cs.DS]
6. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. In: Munro, I., Wagner, D. (eds.) Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX 2008), pp. 13–26. SIAM, Philadelphia (2008)
7. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. Submitted to the ACM Journal of Experimental Algorithmics (2008) (full paper)
8. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 303–318. Springer, Heidelberg (2008)
9. Bauer, R., Delling, D., Wagner, D.: Experimental Study on Speed-Up Techniques for Timetable Information Systems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007), pp. 209–225. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
10. Bingmann, T.: Visualisierung sehr großer Graphen. Student Research Project (2006)
11. Cooke, K., Halsey, E.: The Shortest Route Through a Network with Time-Dependent Intermodal Transit Times. Journal of Mathematical Analysis and Applications (14), 493–498 (1966)
12. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1962)
13. Dean, B.C.: Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
14. Delling, D.: Time-Dependent SHARC-Routing. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 332–343. Springer, Heidelberg (2008)
15. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Routing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)

16. Delling, D., Nannicini, G.: Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks. In: Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008). LNCS. Springer, Heidelberg (2008) (to appear)
17. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: Demetrescu, et al. (eds.) [19]
18. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 52–65. Springer, Heidelberg (2007)
19. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): 9th DIMACS Implementation Challenge - Shortest Paths (November 2006)
20. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1, 269–271 (1959)
21. Disser, Y., Müller-Hannemann, M., Schnee, M.: Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 347–361. Springer, Heidelberg (2008)
22. Fakcharoenphol, J., Rao, S.: Planar Graphs, Negative Weight Edges, Shortest Paths, and near Linear Time. Journal of Computer and System Sciences 72(5), 868–889 (2006)
23. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
24. Goldberg, A.: Personal communication (2008)
25. Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A* Search Meets Graph Theory. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 156–165 (2005)
26. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006), pp. 129–143. SIAM, Philadelphia (2006)
27. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Better Landmarks Within Reach. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 38–51. Springer, Heidelberg (2007)
28. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Shortest Path Algorithms with Preprocessing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
29. Goldberg, A.V., Werneck, R.F.: Computing Point-to-Point Shortest Paths from External Memory. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX 2005), pp. 26–40. SIAM, Philadelphia (2005)
30. Gunkel, T., Müller-Hannemann, M., Schnee, M.: Improved Search for Night Train Connections. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007), pp. 243–258. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
31. Gutman, R.J.: Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 100–111. SIAM, Philadelphia (2004)

32. Hart, P.E., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107 (1968)
33. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear)
34. Holzer, M.: Engineering Planar-Separator and Shortest-Path Algorithms. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2008)
35. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006). SIAM, Philadelphia (2006)
36. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. ACM Journal of Experimental Algorithmics (2008) (to appear)
37. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining Speed-up Techniques for Shortest-Path Computations. ACM Journal of Experimental Algorithmics 10 (2006)
38. Holzer, M., Schulz, F., Willhalm, T.: Combining Speed-up Techniques for Shortest-Path Computations. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 269–284. Springer, Heidelberg (2004)
39. Ishikawa, K., Ogawa, M., Azuma, M., Ito, T.: Map Navigation Software of the Electro-Multivision of the 91 Toyoto Soarer. In: Proceedings of the Vehicle Navigation and Information Systems Conference (VNIS 1991), pp. 463–473. IEEE Computer Society, Los Alamitos (1991)
40. Kaufman, D.E., Smith, R.L.: Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. Journal of Intelligent Transportation Systems 1(1), 1–11 (1993)
41. Klein, P.N.: Multiple-Source Shortest Paths in Planar Graphs. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 146–155 (2005)
42. Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 36–45. SIAM, Philadelphia (2007)
43. Köhler, E., Möhring, R.H., Schilling, H.: Acceleration of Shortest Path and Constrained Shortest Path Computation. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 126–138. Springer, Heidelberg (2005)
44. Lauther, U.: An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, vol. 22, pp. 219–230. IfGI prints (2004)
45. Lauther, U.: An Experimental Evaluation of Point-To-Point Shortest Path Calculation on Roadnetworks with Precalculated Edge-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear)
46. Maue, J., Sanders, P., Matijevic, D.: Goal Directed Shortest Path Queries Using Precomputed Cluster Distances. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 316–327. Springer, Heidelberg (2006)

47. Meyer, U.: Single-Source Shortest-Paths on Arbitrary Directed Graphs in Linear Average-Case Time. In: Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 797–806 (2001)
48. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speed Up Dijkstra's Algorithm. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 189–202. Springer, Heidelberg (2005)
49. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speedup Dijkstra's Algorithm. ACM Journal of Experimental Algorithmics 11, 2.8 (2006)
50. Müller, K.: Berechnung kürzester Pfade unter Beachtung von Abbiegeverboten. Student Research Project (2005)
51. Müller, K.: Design and Implementation of an Efficient Hierarchical Speed-up Technique for Computation of Exact Shortest Paths in Graphs. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik (June 2006)
52. Muller, L.F., Zachariasen, M.: Fast and Compact Oracles for Approximate Distances in Planar Graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 657–668. Springer, Heidelberg (2007)
53. Müller-Hannemann, M., Schnee, M.: Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 246–263. Springer, Heidelberg (2007)
54. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable Information: Models and Algorithms. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 67–90. Springer, Heidelberg (2007)
55. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* Search for Time-Dependent Fast Paths. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 334–346. Springer, Heidelberg (2008)
56. Orda, A., Rom, R.: Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. Journal of the ACM 37(3), 607–625 (1990)
57. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. ACM Journal of Experimental Algorithmics 12, Article 2.4 (2007)
58. Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005)
59. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 804–816. Springer, Heidelberg (2006)
60. Sanders, P., Schultes, D.: Engineering Fast Route Planning Algorithms. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 23–36. Springer, Heidelberg (2007)
61. Sanders, P., Schultes, D.: Robust, Almost Constant Time Shortest-Path Queries in Road Networks. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
62. Sanders, P., Schultes, D., Vetter, C.: Mobile Route Planning. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 732–743. Springer, Heidelberg (2008)
63. Schilling, H.: Route Assignment Problems in Large Networks. Ph.D thesis, Technische Universität Berlin (2006)
64. Schultes, D.: Route Planning in Road Networks. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (February 2008)

65. Schultes, D., Sanders, P.: Dynamic Highway-Node Routing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 66–79. Springer, Heidelberg (2007)
66. Schulz, F.: Timetable Information and Shortest Paths. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2005)
67. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. In: Vitter, J.S., Zaroliagis, C.D. (eds.) WAE 1999. LNCS, vol. 1668, pp. 110–123. Springer, Heidelberg (1999)
68. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. ACM Journal of Experimental Algorithmics 5 (2000)
69. Schulz, F., Wagner, D., Zaroliagis, C.: Using Multi-Level Graphs for Timetable Information in Railway Systems. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 43–59. Springer, Heidelberg (2002)
70. Thorup, M.: Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001), pp. 242–251. IEEE Computer Society Press, Los Alamitos (2001)
71. Thorup, M.: Integer Priority Queues with Decrease Key in Constant Time and the Single Source Shortest Paths Problem. In: Proceedings of the 35th Annual ACM Symposium on the Theory of Computing (STOC 2003), June 2003, pp. 149–158 (2003)
72. U.S. Census Bureau, Washington, DC. UA Census 2000 TIGER/Line Files (2002), `http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html`
73. Volker, L.: Route planning in road networks with turn costs. Studienarbeit, Universität Karlsruhe, Institut für theoretische Informatik (2008)
74. Wagner, D., Willhalm, T.: Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 776–787. Springer, Heidelberg (2003)
75. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest-Path Computation. ACM Journal of Experimental Algorithmics 10, 1.3 (2005)
76. Willhalm, T.: Engineering Shortest Paths and Layout Algorithms for Large Graphs. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2005)