# Combining Speed-up Techniques for Shortest-Path Computations

Martin Holzer* Frank Schulz* Dorothea Wagner* Thomas Willhalm*

March 9, 2006

**Abstract**

Computing a shortest path from one node to another in a directed graph is a very common task in practice. This problem is classically solved by Dijkstra's algorithm. Many techniques are known to speed up this algorithm heuristically, while optimality of the solution can still be guaranteed. In most studies, such techniques are considered individually. The focus of our work is *combination* of speed-up techniques for Dijkstra's algorithm. We consider all possible combinations of four known techniques, namely *goal-directed search*, *bidirectional search*, *multi-level approach*, and *shortest-path containers*, and show how these can be implemented. In an extensive experimental study we compare the performance of the various combinations and analyze how the techniques harmonize when applied jointly. Several real-world graphs from road maps and public transport and three types of generated random graphs are taken into account.

## 1 Introduction

We consider the problem of (repeatedly) finding single-source single-target shortest paths in large, sparse graphs. Typical applications of this problem include route planning systems for cars, bikes, and hikers [Zhan and Noon, 2000, Barrett et al., 2002] or scheduled vehicles like trains and buses [Nachtigall, 1995, Preuss and Syrbe, 1997], spatial databases [Shekhar et al., 1997], and web searching [Barrett et al., 2000]. Usually, the problem is solved by Dijkstra's algorithm [Dijkstra, 1959], which is a label-setting single-source algorithm and as such can be terminated once the target is reached. Besides Dijkstra's algorithm, with a worst-case running time of $\mathcal{O}(m+n\log n)$ using Fibonacci heaps [Fredman and Tarjan, 1987], there are many recent algorithms that solve variants and special cases of the shortest-path problem with better running time (worst-case or average-case; see [Cherkassky et al., 1996] for an experimental comparison, [Zwick, 2001] for a survey, and [Goldberg, 2001b, Meyer, 2001, Pettie et al., 2002] for some more recent work).

The focus of this paper are variants of Dijkstra's algorithm—also denoted as speed-up techniques in the following—that further exploit the fact that a target is given. Typically, such improvements of Dijkstra's algorithm cannot be proven to be asymptotically faster than the original algorithm, and are heuristics in this sense. However, it can be empirically shown that they indeed improve the running time drastically for many realistic data sets. During

*Address: Universität Karlsruhe (TH), Fakultät für Informatik, Postfach 6980, 76128 Karlsruhe, Germany. Email: {mholzer,fschulz,dwagner,willhalm}@ira.uka.de.

the last few years, several new techniques of that kind have been developed. In the scenario described above, it is affordable, and applied by most of these new techniques, to precompute and store additional information on shortest paths, which is used in the on-line phase to reduce the running time for solving a shortest-path query. In [Willhalm and Wagner, 2006], a survey on such speed-up techniques for Dijkstra's algorithm is provided (see also [Willhalm, 2005]). For our study, we exemplarily consider the following four speed-up techniques:

**Goal-Directed Search.** The given edge weights are modified to favor edges leading towards the target node [Hart et al., 1968, Shekhar et al., 1993]. With graphs from timetable information, a speed-up in running time of a factor of roughly 1.5 is reported in [Schulz et al., 2000].

**Bidirectional Search.** Start a second search backwards, from the target to the source (see [Ahuja et al., 1993], Section 4.5). Both searches stop when their search horizons meet. Experiments in [Pohl, 1969] showed that search space can be reduced by a factor of 2, and in [Kaindl and Kainz, 1997] it was shown that combinations with goal-directed search can be beneficial.

**Multi-Level Approach.** This approach takes advantage of hierarchical coarsenings of the given graph, where additional edges have to be computed. These can be regarded as distributed to multiple levels. Depending on the given query, only a small fraction of these edges have to be considered to find a shortest path. Using this technique, speed-up factors of more than 3.5 were observed for road map and public-transport graphs [Holzer, 2003]. Timetable information queries could be improved by a factor of 11 (see [Schulz et al., 2002]), and also in [Jung and Pramanik, 2002] good improvement for road maps is reported.

**Shortest-Path Containers.** These containers provide a necessary condition for each edge, whether or not it has to be respected during the search. More precisely, the bounding box of all nodes that can be reached on a shortest path using this edge is stored. Speed-up factors in the range between 10 and 20 can be achieved [Wagner and Willhalm, 2003].

Our main focus is *combination* of these speed-up techniques. We selected only four techniques in order to provide a detailed analysis of all possible combinations (the complexity of the analysis grows exponentially with the number of techniques under investigation). However, we believe that hereby we cover the characteristics of most of the existing approaches: Goal-directed search and shortest-path containers, as several other approaches, too, are only applicable if a layout of the graph is provided, and multi-level approach and shortest-path containers both require a preprocessing, calculating additional edges and containers, respectively.

The combination of the four techniques is very natural, since all of the techniques modify the search space of Dijkstra's algorithm independently of each other: Goal-directed search directs the search space towards the target of the search by modifying the edge lengths; bidirectional search maintains two search spaces; the multi-level graph approach runs common Dijkstra's algorithm on a subgraph of the augmented input graph; and with shortest-path containers, search space can be pruned by ignoring such edges that for sure do not contribute to a shortest path.

The main question is whether the search space of a combination is better (i.e., smaller) than the one of a single speed-up technique. Another point concerns the additional effort

needed to reduce the search space. For example, with goal-directed search, edge weights have to be calculated during the search. This additional effort usually increases the running time per visited edge by a small constant factor. Considering a combination of techniques, the constant factor per edge is higher than with a single technique, so there is a trade-off between reduction of search space and the additional running time per edge in the search space.

The single techniques are, as mentioned above, heuristics in the sense that the reduction of the search space cannot be proven in general. Hence, the same holds in particular for a combination of such techniques, and the method of choice to answer the questions posed above is an extensive experimental study of the combinations. Since even the single speed-up techniques do not work equally well on all kinds of graphs, we consider several types of both real-world and randomly generated graphs.

The next section contains, after some definitions, a description of the speed-up techniques and shows in more detail how to combine them. Section 3 presents the experimental setup and data sets for the statistics, and the belonging results are given in Section 4. Section 5, finally, gives some conclusions.

# 2 Definitions and Problem Description

## 2.1 Definitions

A directed simple *graph* $G$ is a pair $(V, E)$, where $V$ is the set of nodes and $E \subseteq V \times V$ the set of edges in $G$. Throughout this paper, the number of nodes, $|V|$, is denoted by $n$ and the number of edges, $|E|$, by $m$.

A *path* in $G$ is a sequence of nodes $u_1, \ldots, u_k$ such that $(u_i, u_{i+1}) \in E$ for all $1 \leq i < k$. Given non-negative edge lengths $l : E \to \mathbb{R}_0^+$, the *length* of a path $u_1, \ldots, u_k$ is the sum of weights of its edges, $\sum_{i=1}^{k-1} l(u_i, u_{i+1})$. The *(single-source single-target) shortest-path problem* consists of finding a path of minimum length from a given source $s \in V$ to a target $t \in V$.

A graph *layout* is a mapping $L : V \to \mathbb{R}^2$ of the graph's nodes to the Euclidean plane. For ease of notation, we will identify a node $v \in V$ with its location $L(v)$ in the plane. The Euclidean distance between two nodes $u, v \in V$ is then denoted by $d(u, v)$.

## 2.2 Speed-up Techniques

Our base algorithm is Dijkstra's algorithm using binary heaps as priority queue. In this section, we provide a short description of the four speed-up techniques, whose combinations are discussed in the next section. For a synopsis of the main features, cf. Table 1.

### 2.2.1 Goal-Directed Search (go)

This technique uses a potential function on the node set. The edge lengths are modified in order to direct the graph search towards the target. Let $\lambda$ be such a potential function. The new length of an edge $(v, w)$ is defined to be $\overline{l}(v, w) := l(v, w) - \lambda(v) + \lambda(w)$. The potential must fulfill the condition that for each edge $e$, its new edge length $\overline{l}(e)$ is non-negative, in order to guarantee optimal solutions.

In case edge lengths are Euclidean distances, the Euclidean distance $d(u, t)$ of a node $u$ to the target $t$ is a valid potential, due to the triangle inequality. Otherwise, a potential function

| | orig | prune | info | pre-time | pre-space |
|---|---|---|---|---|---|
| `go` | (✓) | — | potential | — | — |
| `bi` | ✓ | — | — | — | — |
| `ml` | — | ✓ | selections | $\mathcal{O}(ln \cdot T(\text{SSSP}))^*$ | $\mathcal{O}(ln^2)^*$ |
| `sc` | ✓ | ✓ | layout | $\mathcal{O}(n \cdot T(\text{SSSP}))$ | $\mathcal{O}(m)$ |

Figure 1: Main features of the speed-up techniques: original graph used for routing, pruning involved, additional information used, preprocessing time, space requirement of precomputed information.

* see [Holzer et al., 2006] for tighter bounds.

can be defined as follows: let $v_{\max}$ denote the maximum "edge-speed" $d(u,v)/l(e)$ over all edges $e = (u,v)$. The potential of a node $u$ can now be defined as $\lambda(u) = d(u,t)/v_{\max}$.

### 2.2.2 Bidirectional Search (`bi`)

Bidirectional search simultaneously applies the "normal," or forward, variant of the algorithm, starting at the source node, and a so-called reverse, or backward, variant of Dijkstra's algorithm, starting at the destination node. With the reverse variant, the algorithm is applied to the reverse graph, i.e., a graph with the same node set $V$ as that of the original graph, and the reverse edge set $\overline{E} = \{(u,v) \mid (v,u) \in E\}$. Let $d_f(u)$ be the distance labels of the forward search and $d_b(u)$ the labels of the backward search. The algorithm can be terminated when one node has been designated to be permanent by both the forward and the reverse algorithm. Then the shortest path is determined by the node $u$ with minimum value $d_f(u) + d_b(u)$ and can be composed of the one from the start node to $u$, found by the forward search, and the edges reverted again on the path from the destination to $u$, found by the reverse search.

### 2.2.3 Multi-Level Approach (`ml`)

This speed-up technique requires a preprocessing step at which the input graph $G = (V,E)$ is decomposed into $l+1$ ($l \geq 1$) levels and enriched with additional edges representing shortest paths between certain nodes. This decomposition depends on subsets $S_i$ of the graph's node set for each level, called selected nodes at level $i$: $S_0 := V \supseteq S_1 \supseteq \ldots \supseteq S_l$. These node sets can be determined on diverse criteria; with our implementation, they consist of the desired numbers of nodes with highest degree in the graph, which has turned out to be an appropriate criterion [Holzer, 2003].

There are three different types of edges being added to the graph: *upward edges*, going from a node that is not selected at one level to a node selected at that level, *downward edges*, going from selected to non-selected nodes, and *level edges*, passing between selected nodes at one level. The weight of such an edge is assigned the length of a shortest path between the end-nodes.

To find a shortest path between two nodes, then, it suffices for Dijkstra's algorithm to consider a relatively small subgraph of the "multi-level graph" (a certain set of upward and of downward edges and a set of level edges passing at a maximal level that has to be taken into account for the given source and target nodes).

### 2.2.4 Shortest-Path Containers (sc)

This speed-up technique requires a preprocessing computing all shortest- path trees. For each edge $e \in E$, we compute the set $S(e)$ of those nodes to which a shortest path starts with edge $e$. Using a given layout, we then store for each edge $e \in E$ the bounding box of $S(e)$ in an associative array $C$ with index set $E$.

It is then sufficient to perform Dijkstra's algorithm on the subgraph induced by the edges $e \in E$ with the target node included in $C[e]$. This subgraph can be determined on the fly, by excluding all other edges in the search. (One can think of shortest-path containers as traffic signs which characterize the region that they lead to.)

A variation of this technique is introduced in [Schulz et al., 2000], where as geometric objects angular sectors instead of bounding boxes were used, for application to a timetable information system. An extensive study in [Wagner and Willhalm, 2003] showed that bounding boxes are the best geometric objects in terms of running time and competitive with much more complex geometric objects in terms of visited nodes.

## 2.3 Combining the Speed-up Techniques

In this section, we first outline the key notion of combining each pair of techniques and motivate afterwards that extending these to combinations including three or all four techniques is not difficult.

### 2.3.1 Goal-Directed Search and Bidirectional Search

Combining goal-directed and bidirectional search is not as obvious as it may seem at first glance. [Pohl, 1969] provides a counter-example for the fact that simple application of a goal-directed search forward and backward yields a wrong termination condition. However, the alternative condition proposed there is shown in [Kaindl and Kainz, 1997] to be quite inefficient, as the search in each direction almost reaches the source of the other direction. This often results in a slower algorithm.

To overcome these deficiencies, we simply use the very same edge weights $\overline{l}(v, w) := l(v, w) - \lambda(v) + \lambda(w)$ for both the forward and the backward search. With these weights, the forward search is directed to the target $t$ and the backward search has no preferred direction, but favors edges that are directed towards $t$. This proceeding always computes shortest paths, as an $s$-$t$-path is shortest independent of whether $l$ or $\overline{l}$ is used for the edge weights.

### 2.3.2 Goal-Directed Search and Multi-Level Approach

As described in Section 2.2.3, the multi-level approach basically determines for each query a subgraph of the multi-level graph on which Dijkstra's algorithm is finally run. The computation of this subgraph does not affect edge lengths and thus goal-directed search can be simply performed on it.

### 2.3.3 Goal-Directed Search and Shortest-Path Containers

Similar to the multi-level approach, the shortest-path containers approach determines for a given query a subgraph of the original graph. Again, edge lengths are irrelevant for the computation of the subgraph and goal-directed search can be applied offhand.

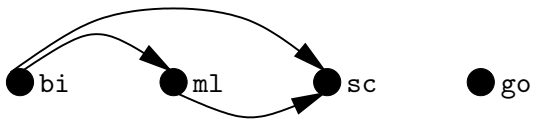


Figure 2: Interdependencies of the speed-up techniques regarding preprocessing.

#### 2.3.4 Bidirectional Search and Multi-Level Approach

Basically, bidirectional search can be applied to the subgraph defined by the multi-level approach. In our implementation, that subgraph is computed on the fly during Dijkstra's algorithm: for each node considered, the set of necessary outgoing edges is determined. To perform a bidirectional search on the multi-level subgraph, a symmetric, backward version of the subgraph computation has to be implemented: for each node considered in the backward search, the incoming edges that are part of the subgraph have to be determined. Shortest paths are guaranteed since bidirectional search is run on a subgraph that preserves optimality and, by the additional edges, only contains supplementary information consistent with the original graph.

#### 2.3.5 Bidirectional Search and Shortest-Path Containers

In order to take advantage of shortest-path containers in both directions of a bidirectional search, a second set of containers is needed. For each edge $e \in E$, we compute the set $S_b(e)$ of those nodes from which a shortest path ending with $e$ exists. We store for each edge $e \in E$ the bounding box of $S_b(e)$ in an associative array $C_b$ with index set $E$. The forward search checks whether the target is contained in $C(e)$, the backward search, whether the source is in $C_b(e)$. It is easy to verify that by construction only such edges are pruned that do not form part of any partial shortest path and thus of any shortest $s$-$t$-path.

#### 2.3.6 Multi-Level Approach and Shortest-Path Containers

The multi-level approach enriches a given graph with additional edges. Each new edge $(u_1, u_k)$ represents a shortest path $(u_1, u_2, \ldots, u_k)$ in $G$. We annotate such a new edge $(u_1, u_k)$ with $C(u_1, u_2)$, the associated bounding box of the first edge on this path. This consistent labeling of new edges, which represent shortcuts in the original graph, ensures still shortest paths.

#### 2.3.7 Extension to Arbitrary Combinations

Extracting the essence from the above discussion, we now motivate that assembling any combination of our speed-up techniques can be done in a straight-forward manner. To this end, we order the techniques in such a way that it reflects their interdependencies when it comes to preprocessing (cf. Figure 2); the actual steps to be taken for a specific combination can then be seen from that ordering.

Bidirectional search requires supplementary information to be precomputed with both the multi-level approach and shortest-path containers, while the multi-level technique entails additional work for shortest-path container preprocessing. Including goal-directed search does not affect preprocessing.

For justification of straight-forwardness of a new *search* algorithm, it suffices to verify that each technique kicks in at a different spot and thus they do not impede one another:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | street | | | | | | | | | |
| $n$ | 1444 | 3045 | 16471 | 20466 | 25982 | 38823 | 45852 | 45073 | 51510 | 79456 |
| $m$ | 3060 | 7310 | 34530 | 42288 | 57620 | 79988 | 98098 | 91314 | 110676 | 172374 |
| | public transport | | | | | | | | | |
| $n$ | 409 | 705 | 1660 | 2279 | 2399 | 4598 | 6884 | 10815 | 12070 | 14335 |
| $m$ | 1215 | 1681 | 4327 | 6015 | 8008 | 14937 | 18601 | 29351 | 33728 | 39887 |
| | planar | | | | | | | | | |
| $n$ | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 |
| $m$ | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| | waxman | | | | | | | | | |
| $n$ | 938 | 1974 | 2951 | 3938 | 4949 | 5946 | 6943 | 7917 | 8882 | 9906 |
| $m$ | 4070 | 9504 | 14506 | 19658 | 24474 | 29648 | 34764 | 39138 | 44208 | 48730 |
| | hierarchical | | | | | | | | | |
| $n$ | 1021 | 2017 | 3014 | 4010 | 5007 | 6002 | 7025 | 8021 | 9016 | 10012 |
| $m$ | 5814 | 10942 | 16138 | 21048 | 26064 | 31030 | 36124 | 41146 | 46246 | 51130 |

Table 1: Number of nodes and edges for all test graphs.

If including bidirectional search, we have to keep track of two search horizons and combine partial shortest paths properly; with the multi-level approach, the search is performed on a subgraph of the enriched graph instead of the original graph; under use of shortest-path containers, some edges may (additionally) be pruned; and performing goal-directed search, the lengths of the edges scanned are modified.

# 3 Experimental Setup

In this section, we provide details on the input data used, consisting of real-world and randomly generated graphs, and on the execution of the experiments.

## 3.1 Data

### 3.1.1 Real-World Graphs

In our experiments we included two sets of graphs that stem from real applications. As in other experimental work, it turned out that using realistic data is quite important, as the performance of the algorithms strongly depends on the characteristics of the data. Note that all our graphs are embedded either by geographic information or by construction.

**Street Graphs.** Our street graphs are networks of U.S. cities and their surroundings. These graphs are bidirected, and edge lengths are Euclidean distances. The graphs are fairly large and very sparse because bends are represented by polygonal lines. (With such a representation of a street network, it is possible to efficiently find the nearest point in a street by a point-to-point search.)

**Public-Transport Graphs.** A public-transport graph represents a network of trains, buses, and other scheduled vehicles. The nodes of such a graph correspond to stations or stops,

and there exists an edge between two nodes if there is a non-stop connection between the respective stations. The weight of an edge is the average travel time of all vehicles that contribute to this edge. In particular, the edge lengths are not Euclidean distances in this set of graphs.

### 3.1.2 Random Graphs

We generated three sets of random graphs: planar, Waxman, and some kind of hierarchical graphs. Each set consists of ten connected, bidirected graphs with (approximately) $n = 1000 \cdot i$ nodes ($i = 1, \ldots, 10$) and an average out-degree of 2.5 (i.e., the graphs have approximately $5n$ edges).

**Random Planar Graphs.** For construction of random planar graphs, we used a generator provided by LEDA [Näher and Mehlhorn, 1999]. A given number, $n$, of nodes are uniformly distributed in a square with a lateral length of 1, and a triangulation of the nodes is computed. This yields a complete undirected planar graph. Finally, edges are deleted at random until the graph contains $2.5 \cdot n$ edges, and each of these is replaced by two directed edges, one in either direction.

**Random Waxman Graphs.** Construction of these graphs is based on a random graph model introduced by Waxman [Waxman, 1988]. We use this model in an attempt to emulate one aspect of our public-transport graphs, the existance of *long-distance edges*, i.e., edges linking rather far-apart nodes. As previous experiments showed, copying node degrees from public-transport graphs does not suffice to obtain random graphs with similar properties.

Input parameters to this model are the number of nodes $n$ and two positive rational numbers $\alpha$ and $\beta$. The nodes are again distributed uniformly in a square of a lateral length of 1, and the probability that an edge $(u, v)$ exists is $\beta \cdot \exp(-d(u, v)/(\sqrt{2}\alpha))$. Higher $\beta$ values increase the edge density, while smaller $\alpha$ values increase the density of short edges in relation to long edges. To ensure connectedness and bidirectedness of the graphs, all nodes that do not belong to the largest connected component are deleted (thus, slightly less than $n$ nodes remain) and the graph is bidirected by insertion of missing reverse edges. We set $\alpha = 0.01$ and empirically determined that setting $\beta = 2.5 \cdot 1620/n$ yields the wanted number of edges.

**Hierarchical Graphs.** Our motivation for including such a type of graph in our experiments was that several real-world graph classes exhibit some kind of hierarchical structure; e.g., public-transport graphs as described above typically consist of few stations having great node degrees, forming a coarse overall network or *backbone*, as will be referred to in the following, while the rest of the nodes are more locally connected and have considerably smaller degrees.

Our generated hierarchical graphs can be regarded as $5 \times 5$ grids, constructed roughly as follows: Pick $n/14$ nodes for the backbone, distribute the remaining nodes uniformly to the 25 grid cells, and associate each backbone node with one cell. Construct a Delaunay-triangulation-based planar graph with an edge factor of 2.5 on the nodes of each cell and the backbone nodes. For each cell, three nodes associated with it are linked to nodes from that cell such that all degrees of backbone nodes exceed the greatest degree

of all other nodes. Again, for each edge the respective back edge is inserted, and the largest connected component of the entire graph thus obtained is finally returned.

### 3.2 Experiments

We implemented all combinations of speed-up techniques as described in Sections 2.2 and 2.3 in C++, using the graph and priority queue data structures of the LEDA library (version 4.4; cf. [Näher and Mehlhorn, 1999]). The code was compiled with the GNU compiler (version 3.3), and the experiments were carried out on an Intel Xeon machine with 2.6 GHz and 2 GB of memory, running Linux (kernel version 2.4).

For each graph and combination, we computed for a set of queries shortest paths, measuring two types of *performance*: the mean values of the *running times* (CPU time in seconds) and the number of nodes inserted in the priority queue (also called *visited nodes* in the following). The queries were chosen at random and the amount of them was determined such that statistical relevance can be guaranteed (see also [Wagner and Willhalm, 2003]).

## 4 Experimental Results

The main outcome of our experimental study is shown in Figures 3 and 4. Further diagrams that we used for our analysis are depicted in Figures 5–9. Each combination is referred to by a 4-tuple of tokens: `go` (goal-directed), `bi` (bidirectional), `ml` (multi-level), `sc` (shortest-path container), and `--` if the respective technique is not used (e.g., `go -- ml sc`). In all figures, the graphs are ordered as listed in Table 1.

The outcomes referring to one graph class with one combination of techniques are summarized in the form of *boxplots*: the box represents the middle 50 percent of the respective values (with the median marked as a small horizontal line), the whiskers, i.e., the upper and lower bounds of the dashed lines, are the extremal outcomes within a range 1.5 times the height of the box above and below the border of the box, respectively, and outliers, i.e., outcomes lying beyond the whiskers, are drawn as circles.

We calculated two different values denoting relative *speed-up*: on the one hand, Figures 3–4 show the speed-up that we achieved compared to plain Dijkstra, i.e., for each combination of techniques the ratio of the performance of plain Dijkstra and the performance of Dijkstra with the specific combination of techniques applied. There are separate figures for the number of nodes and running time. The horizontal line in each of these as well as the remaining diagrams marks the border between gain and loss in efficiency relative to plain Dijkstra.

On the other hand, for each of the Figures 5–8, we focus on one technique $\mathcal{T}$ and show for each combination containing $\mathcal{T}$ the speed-up (with respect to the number of visited nodes) that can be achieved compared to the combination without $\mathcal{T}$. For example, when focusing on bidirectional search and considering the combination `go bi -- sc`, say, we investigate by which factor the performance gets better when the combination `go bi -- bb` is used instead of `go -- -- sc` only.

In the following, we discuss, for each technique separately, how combinations with that technique behave, and then turn to the relation of the two performance parameters measured, the number of visited nodes and running time: We define the *overhead* of a combination to be the ratio of running time and the number of visited nodes. In other words, the overhead reflects the time spent per node.

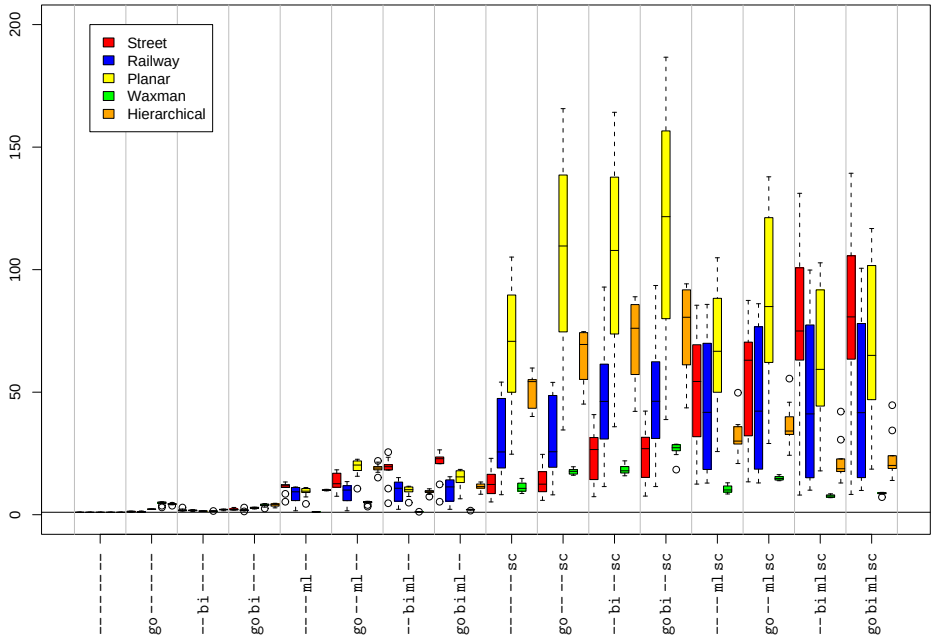


Figure 3: Speed-up relative to Dijkstra's algorithm in terms of visited nodes.

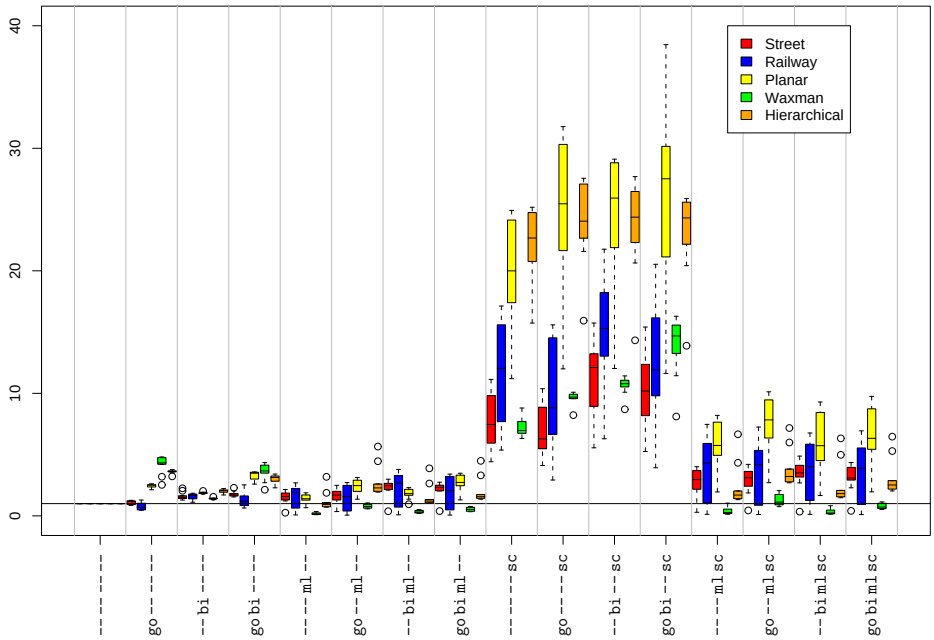


Figure 4: Speed-up relative to Dijkstra's algorithm in terms of running time.

## 4.1 Speed-up of the Combinations

### 4.1.1 Goal-Directed Search

Comparing the number of nodes visited with pure goal-directed search to that with plain Dijkstra, speed-up varies a lot between the different types of graphs (cf. Figures 3 and 5): We get a speed-up of about 2 for the planar graphs, and of 4 to 5 for the hierarchical and Waxman graphs, which is surprisingly good compared to speed-up values of less than 2 observed with the real-world graphs.

Adding goal-directed search to the multi-level approach performs a little worse than adding it to plain Dijkstra and with bidirectional search, we get another slight deterioration. Adding it to shortest-path containers is hardly beneficial (see also Figure 5).

Figure 4 reveals that, for real-world graphs, adding goal-directed search to any combination does not improve the running time. For generated graphs, however, for most combinations running time decreases when goal-directed search is applied additionally. In particular, it is advantageous to add it to a combination containing the multi-level approach. To conclude, our experiments indicate that combining goal-directed search with the multi-level approach is generally a good idea.

### 4.1.2 Bidirectional Search

Pure bidirectional search yields a speed-up of between 1.5 and 2 for the number of visited nodes (cf. Figure 3) and for the running time (cf. Figure 4), for all types of graphs (for the hierarchical graphs, the speed-up, of a factor of more than 2, is even better).

For combinations of bidirectional search with other speed-up techniques, the situation is different and depends heavily on the type of graph, as Figure 6 shows: For street graphs, bidirectional search almost always gives an actual speed-up compared to any combination of techniques without bidirectional search. For Waxman and hierarchical graphs, almost none of the combinations can be improved by additional application of bidirectional search. Only shortest-path containers and the combination of shortest-path containers with goal-directed search can always be improved through bidirectional search.

### 4.1.3 Multi-Level Approach

The multi-level approach crucially depends on the decomposition of the given graph, i.e., the balancedness of the resulting multi-level graph. The Waxman graphs could not be decomposed properly and therefore all combinations containing the multi-level approach yield speed-up factors of less than 1 for the Waxman graphs, which means a slowing down (cf. Figure 4).

Thus, we consider only the remaining graph classes: with these graphs, pure multi-level approach reduces the number of visited nodes to a similar extent, the median values of the observed speed-up factors (cf. Figure 7) ranging between 9 and 12 for all graph classes, which implies good decomposability, as required for the approach. Note that large ranges of speed-up factors inside one graph class can be observed (consider, e.g., the public-transport graphs in Figure 7). Reasons for this behavior are on the one hand the fact that the achieved decompositions are not always of similar quality within one graph class (e.g., the French railway network, with Paris as its center, is very different from the German railway network, which is more evenly spread); on the other hand, the speed-up achieved with the multi-level approach also depends on the size of the graph.

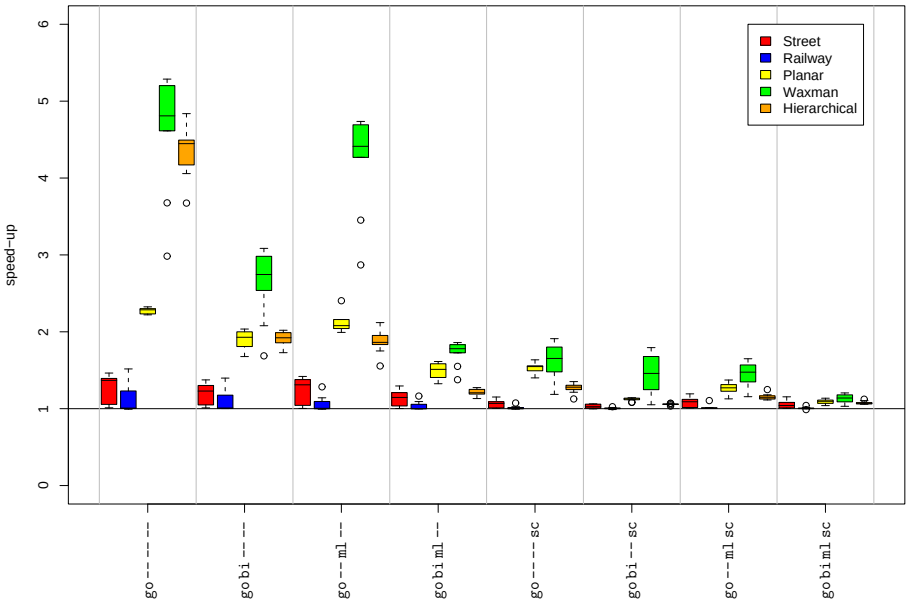


Figure 5: Speed-up relative to the respective combination without goal-directed search in terms of visited nodes.

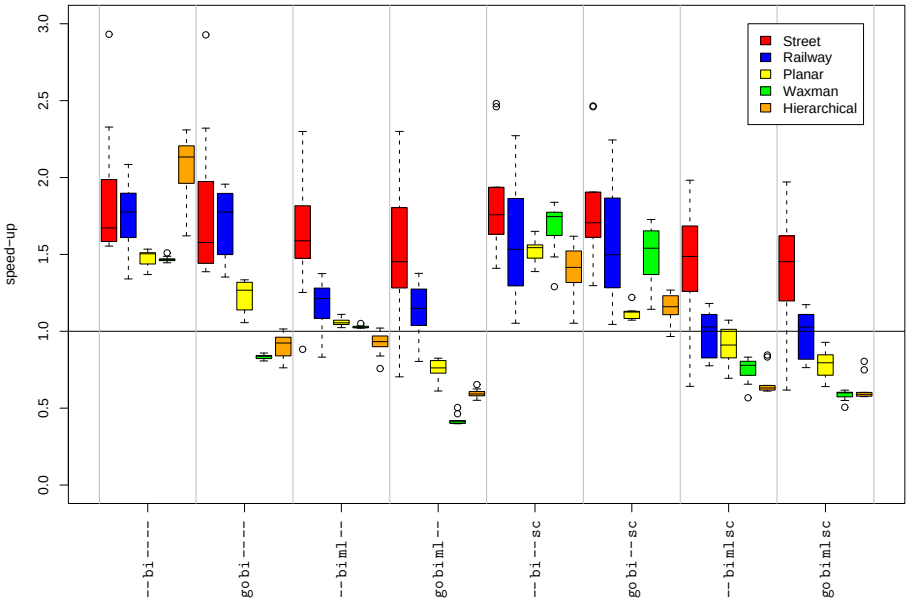


Figure 6: Speed-up relative to the respective combination without bidirectional search in terms of visited nodes.

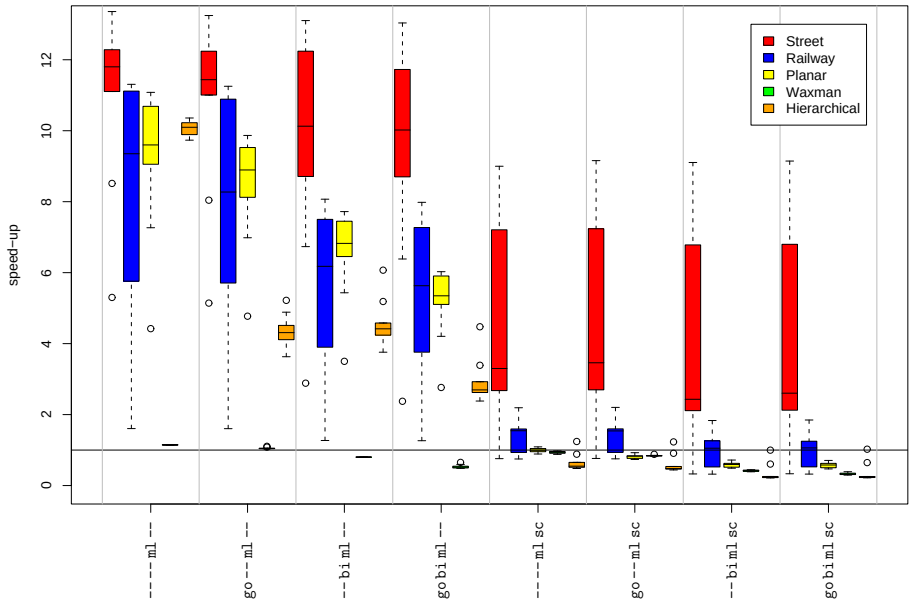


Figure 7: Speed-up relative to the respective combination without multi-level approach in terms of visited nodes.

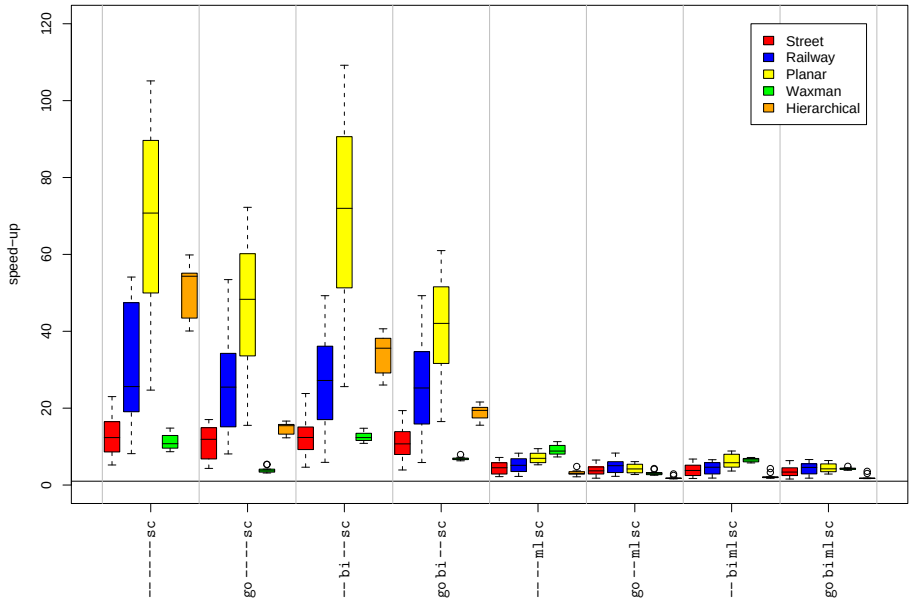


Figure 8: Speed-up relative to the respective combination without shortest-path containers in terms of visited nodes.

Adding the multi-level approach to goal-directed and bidirectional search and their combination also results in good improvement for the number of nodes (cf. Figure 7). In combination with shortest-path containers, the multi-level approach is beneficial only in the case of street graphs.

Caused by the big overhead of the multi-level approach (cf. Figure 9), however, running time cannot be improved in the same order of magnitude as the number of nodes (cf. Figure 4). Also, the multi-level approach allows tuning of several parameters, such as the number of levels and the choice of the selected nodes. This tuning crucially depends on the input graph [Holzer, 2003, Holzer et al., 2006]. Hence, we believe that considerable improvement of the presented results is possible if specific parameters are chosen for every single graph.

#### 4.1.4 Shortest-Path Containers

Shortest-path containers work especially well when applied to planar graphs (see Figure 8); actually, speed-up even increases with the size of the graph (which again yields large ranges of speed-up factors within one graph class, as similarly observed with the multi-level approach). For Waxman graphs, the situation is completely different: with graph size, speed-up gets smaller (not shown in the diagrams). This can be explained by the fact that large Waxman graphs have, due to construction, more long-distance edges than small ones. Because of this, shortest paths become more tortuous and the bounding boxes contain more "wrong" nodes.

Figures 3, 4, and 8 show that throughout the different types of graphs, shortest-path containers individually as well as in combination with goal-directed and bidirectional search yield exceptionally high speed-ups. Only the combinations that include the multi-level approach cannot be improved that much.

### 4.2 Overhead

For goal-directed and bidirectional search, the overhead (time per visited node) is quite small, while for shortest-path containers it is of a factor of about 2 compared to plain Dijkstra (cf. Figure 9). The overhead caused by the multi-level approach is generally high and varies quite a lot depending on the type of graph. As Waxman graphs do not decompose well, the overhead for the multi-level approach is large and becomes even larger when the size of the graph increases. For very large street graphs, the multi-level approach overhead increases dramatically. We assume that it would be necessary to add another level for graphs of this size.

It is also interesting to note that the relative overhead of the combination `go bi ml --` is smaller than that of pure multi-level—especially for the generated graphs.

## 5 Conclusion and Outlook

To summarize, we conclude that there are speed-up techniques that combine well and others where speed-up does not scale. Our main result is that goal-directed search and multi-level approach is a good combination and bidirectional search and shortest-path containers complement each other.

For real-world graphs, the combination `-- bi ml sc` is the best choice as to the number of visited nodes. In terms of running time, the winner is `-- bi -- sc`. For generated graphs, the best combination is `go bi -- sc` for both the number of nodes and running time.

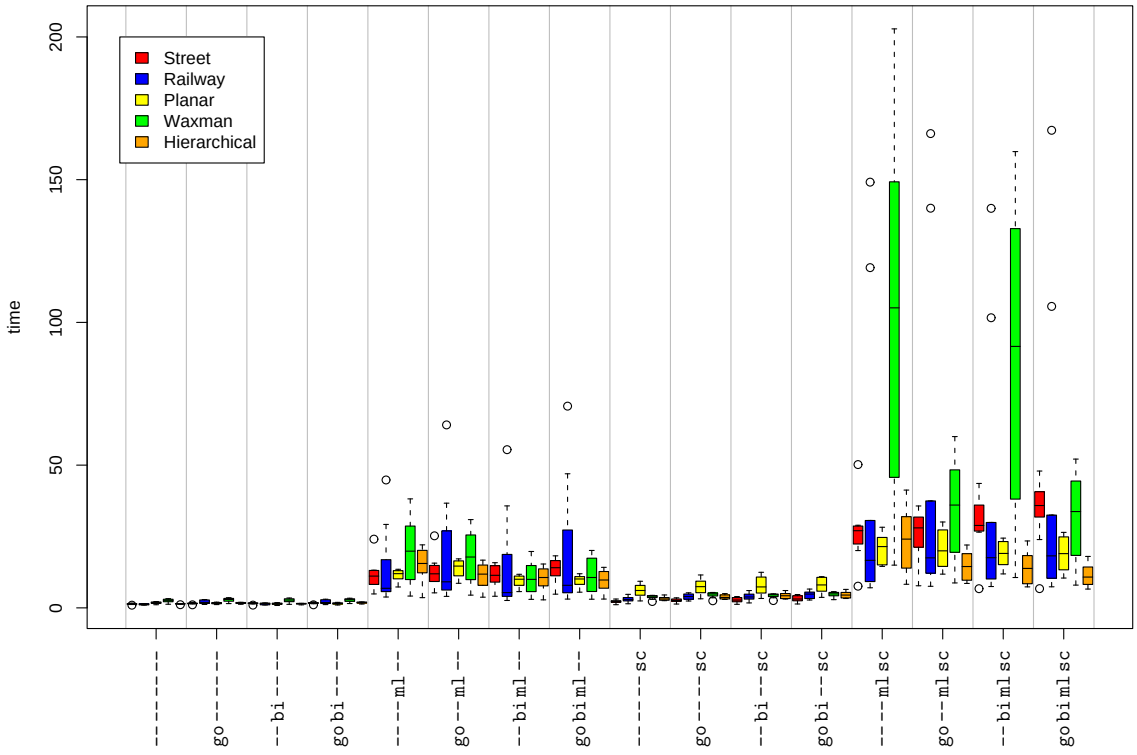


Figure 9: Average running time per visited node in $\mu s$.

When waiving an expensive preprocessing, the combination `go bi ---` is generally the fastest heuristic with smallest search space—except for Waxman graphs, for which pure goal-directed search is better. Actually, goal-directed search is the only speed-up technique that works comparatively well for Waxman graphs. Because of this specific behavior, we conclude that the planar and hierarchical graphs are a better approximation of the real-world graphs than the Waxman graphs, which is also confirmed by the fact that with the multi-level approach, all graphs except the Waxman graphs exhibit similar behavior.

It would be interesting to further investigate combinations with improved variants of the approaches considered in this paper (e.g., in [Holzer et al., 2006] speed-up factors of up to 50 have been observed with road graphs in the multi-level approach) and other, very recently developed speed-up techniques relying on preprocessed information (which have already been mentioned briefly in the introduction). The relation of the four techniques under investigation in this work to those other approaches is as follows:

(i) The *ALT* algorithm proposed in [Goldberg and Harrelson, 2005] improves goal-directed search by precomputed shortest-path information from and to so-called "landmark nodes". The combination of this algorithm with bidirectional search was investigated in the same work.

(ii) Approaches relying on hierarchical decomposition of the underlying graph by means of *edge-separators* [Jung and Pramanik, 2002, Flinsenberg, 2004] are often applied to road networks, where the edge separators are usually determined by topographical information. Another approach relying on graph decompositions are the so-called *hierarchically encoded path views* [Jing et al., 1998]. These techniques are very similar to the multi-level approach.

(iii) Techniques to divide a graph into regions and to store those very regions reachable via an edge in *bit-vectors* attached to that edge [Köhler et al., 2004, Lauther, 2004] are closely related to the shortest-path containers.

(iv) In the *reach-based routing* scheme [Gutman, 2004], for each vertex a "reach value" (which intuitively reflects the lengths of shortest paths on which it lies) is computed beforehand and used during the online phase to speed up the shortest-path search. The combination of reach-based routing with goal-directed search has been shown to be fruitful [Goldberg et al., 2006]. In the latter work, also several improvements of the reach-based routing are presented.

(v) Finally, we would like to mention a speed-up technique that precomputes *highway hierarchies* [Sanders and Schultes, 2005]. In the online phase, depending on the distance from the source and to the target, edges in lower levels of the hierarchy can be ignored. This technique relies on a bidirectional variant of Dijkstra's algorithm and is related to a variant of reach-based routing that uses reach values for edges (cf. [Goldberg et al., 2006]).

Another starting point for future work is given by the observation that, except for bidirectional search, the speed-up techniques under investigation in this work can be regarded as a standard run of Dijkstra's algorithm on a modified graph. From a shortest path in the modified graph one can easily determine a shortest path in the original graph. It is an interesting question whether the techniques can be applied directly, or in a modified fashion, to improve also the running time of other shortest-path algorithms.

Furthermore, specialized priority queues used in Dijkstra's algorithm have been shown to be fast in practice [Dial, 1969, Goldberg, 2001a]. Using such queues would provide the same results for the number of visited nodes. Running times, however, would be different and therefore interesting to evaluate.

## References


[Ahuja et al., 1993] Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows*. Prentice–Hall.

[Barrett et al., 2002] Barrett, C., Bisset, K., Jacob, R., Konjevod, G., and Marathe, M. (2002). Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In *Proc. 10th European Symposium on Algorithms (ESA)*, volume 2461 of *LNCS*, pages 126–138. Springer.

[Barrett et al., 2000] Barrett, C., Jacob, R., and Marathe, M. (2000). Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837.

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

[Dial, 1969] Dial, R. (1969). Algorithm 360: Shortest path forest with topological ordering. *Communications of ACM*, 12:632–633.

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

[Flinsenberg, 2004] Flinsenberg, I. C. (2004). *Route Planning Algorithms for Car Navigation*. PhD thesis, Technische Universiteit Eindhoven.

[Fredman and Tarjan, 1987] Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615.

[Goldberg and Harrelson, 2005] Goldberg, A. and Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*. SIAM.

[Goldberg et al., 2006] Goldberg, A., Kaplan, H., and Werneck, R. (2006). Reach for A*: Efficient point-to-point shortest path algorithms. In *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX06)*. SIAM. To appear in the same volume.

[Goldberg, 2001a] Goldberg, A. V. (2001a). Shortest path algorithms: Engineering aspects. In *Proc. International Symposium on Algorithms and Computation (ISAAC)*, volume 2223 of *LNCS*, pages 502–513. Springer.

[Goldberg, 2001b] Goldberg, A. V. (2001b). A simple shortest path algorithm with linear average time. In *Proc. 9th European Symposium on Algorithms (ESA)*, volume 2161 of *LNCS*, pages 230–241. Springer.

[Gutman, 2004] Gutman, R. J. (2004). Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proc. 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 100–111. SIAM.

[Hart et al., 1968] Hart, P., Nilsson, N. J., and Raphael, B. A. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. Cybernet.*, 2.

[Holzer, 2003] Holzer, M. (2003). Hierarchical speed-up techniques for shortest-path algorithms. Technical report, Dept. of Informatics, University of Konstanz, Germany. http://www.ub.uni-konstanz.de/kops/volltexte/2003/1038/.

[Holzer et al., 2006] Holzer, M., Schulz, F., and Wagner, D. (2006). Engineering multi-level overlay graphs for shortest-path queries. In *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. To appear.

[Jing et al., 1998] Jing, N., Huang, Y.-W., and Rundensteiner, E. A. (1998). Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Trans. Knowledge and Data Engineering*, 10(3).

[Jung and Pramanik, 2002] Jung, S. and Pramanik, S. (2002). An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1029–1046.

[Kaindl and Kainz, 1997] Kaindl, H. and Kainz, G. (1997). Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7:283–317.

[Köhler et al., 2004] Köhler, E., Möhring, R. H., and Schilling, H. (2004). Acceleration of shortest path computation. Article 42, Technische Universität Berlin, Fakultät II Mathematik und Naturwissenschaften.

[Lauther, 2004] Lauther, U. (2004). 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*, volume 22, pages 219–230. IfGI prints, Institut für Geoinformatik, Münster.

[Meyer, 2001] Meyer, U. (2001). Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In *Proc. 12th Symp. on Discrete Algorithms*, pages 797–806.

[Nachtigall, 1995] Nachtigall, K. (1995). Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166.

[Näher and Mehlhorn, 1999] Näher, S. and Mehlhorn, K. (1999). *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press. (http://www.algorithmic-solutions.com).

[Pettie et al., 2002] Pettie, S., Ramachandran, V., and Sridhar, S. (2002). Experimental evaluation of a new shortest path algorithm. In *Proc. Algorithm Engineering and Experiments (ALENEX)*, volume 2409 of *LNCS*, pages 126–142. Springer.

[Pohl, 1969] Pohl, I. (1969). Bi-directional and heuristic search in path problems. Technical Report 104, Stanford Linear Accelerator Center, Stanford, California.

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

[Sanders and Schultes, 2005] Sanders, P. and Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. In *Proceedings 17th European Symposium on Algorithms (ESA)*.

[Schulz et al., 2000] Schulz, F., Wagner, D., and Weihe, K. (2000). Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Exp. Algorithmics*, 5(12).

[Schulz et al., 2002] Schulz, F., Wagner, D., and Zaroliagis, C. (2002). Using multi-level graphs for timetable information in railway systems. In *Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX)*, volume 2409 of *LNCS*, pages 43–59. Springer.

[Shekhar et al., 1997] Shekhar, S., Fetterer, A., and Goyal, B. (1997). Materialization trade-offs in hierarchical shortest path algorithms. In *Proc. Symp. on Large Spatial Databases*, pages 94–111.

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

[Wagner and Willhalm, 2003] Wagner, D. and Willhalm, T. (2003). Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *Proc. 11th European Symposium on Algorithms (ESA)*, volume 2832 of *LNCS*, pages 776–787. Springer.

[Waxman, 1988] Waxman, B. M. (1988). Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9).

[Willhalm, 2005] Willhalm, T. (2005). *Engineering Shortest Paths and Layout Algorithms for Large Graphs*. PhD thesis, Universität Karlsruhe (TH), Fakultät Informatik.

[Willhalm and Wagner, 2006] Willhalm, T. and Wagner, D. (2006). Shortest path speedup techniques. In *Algorithmic Methods for Railway Optimization*, LNCS. Springer. To appear.

[Zhan and Noon, 2000] Zhan, F. B. and Noon, C. E. (2000). A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. *Journal of Geographic Information and Decision Analysis*, 4(2).

[Zwick, 2001] Zwick, U. (2001). Exact and approximate distances in graphs - a survey. In *Proc. 9th European Symposium on Algorithms (ESA)*, LNCS, pages 33–48. Springer.