

Route Planning in Transportation Networks

Hannah Bast¹, Daniel Delling², Andrew Goldberg³,
Matthias Müller-Hannemann⁴, Thomas Pajor⁵(✉), Peter Sanders⁶,
Dorothea Wagner⁶, and Renato F. Werneck³

¹ University of Freiburg, Freiburg im Breisgau, Germany
`bast@informatik.uni-freiburg.de`

² Apple Inc., Cupertino, USA
`ddelling@apple.com`

³ Amazon, Seattle, USA
`{andgold,werneck}@amazon.com`

⁴ Martin-Luther-Universität Halle-Wittenberg, Halle, Germany
`muellerh@informatik.uni-halle.de`

⁵ Microsoft Research, Mountain View, USA
`microsoft@tpajor.com`

⁶ Karlsruhe Institute of Technology, Karlsruhe, Germany
`{sanders,dorothea.wagner}@kit.edu`

Abstract. We survey recent advances in algorithms for route planning in transportation networks. For road networks, we show that one can compute driving directions in milliseconds or less even at continental scale. A variety of techniques provide different trade-offs between preprocessing effort, space requirements, and query time. Some algorithms can answer queries in a fraction of a microsecond, while others can deal efficiently with real-time traffic. Journey planning on public transportation systems, although conceptually similar, is a significantly harder problem due to its inherent time-dependent and multicriteria nature. Although exact algorithms are fast enough for interactive queries on metropolitan transit systems, dealing with continent-sized instances requires simplifications or heavy preprocessing. The multimodal route planning problem, which seeks journeys combining schedule-based transportation (buses, trains) with unrestricted modes (walking, driving), is even harder, relying on approximate solutions even for metropolitan inputs.

1 Introduction

This survey is an introduction to the state of the art in the area of practical algorithms for routing in transportation networks. Although a thorough survey by Delling et al. [94] has appeared fairly recently, it has become outdated due to significant developments in the last half-decade. For example, for continent-sized

This work was mostly done while the authors Daniel Delling, Andrew Goldberg, and Renato F. Werneck were at Microsoft Research Silicon Valley.

road networks, newly-developed algorithms can answer queries in a few hundred nanoseconds; others can incorporate current traffic information in under a second on a commodity server; and many new applications can now be dealt with efficiently. While Delling et al. focused mostly on road networks, this survey has a broader scope, also including schedule-based public transportation networks as well as multimodal scenarios (combining schedule-based and unrestricted modes).

Section 2 considers shortest path algorithms for static networks; although it focuses on methods that work well on road networks, they can be applied to arbitrary graphs. Section 3 then considers the relative performance of these algorithms on real road networks, as well as how they can deal with other transportation applications. Despite recent advances in routing in road networks, there is still no “best” solution for the problems we study, since solution methods must be evaluated according to different measures. They provide different trade-offs in terms of query times, preprocessing effort, space usage, and robustness to input changes, among other factors. While solution quality was an important factor when comparing early algorithms, it is no longer an issue: as we shall see, all current state-of-the-art algorithms find provably exact solutions. In this survey, we focus on algorithms that are not clearly dominated by others. We also discuss approaches that were close to the dominance frontier when they were first developed, and influenced subsequent algorithms.

Section 4 considers algorithms for journey planning on schedule-based public transportation systems (consisting of buses, trains, and trams, for example), which is quite different from routing in road networks. Public transit systems have a time-dependent component, so we must consider multiple criteria for meaningful results, and known preprocessing techniques are not nearly as effective. Approximations are thus sometimes still necessary to achieve acceptable performance. Advances in this area have been no less remarkable, however: in a few milliseconds, it is now possible to find good journeys within public transportation systems at a very large scale.

Section 5 then considers a true *multimodal* scenario, which combines schedule-based means of transportation with less restricted ones, such as walking and cycling. This problem is significantly harder than its individual components, but reasonable solutions can still be found.

A distinguishing feature of the methods we discuss in this survey is that they quickly made real-life impact, addressing problems that need to be solved by interactive systems at a large scale. This demand facilitated technology transfer from research prototypes to practice. As our concluding remarks (Sect. 6) will explain, several algorithms we discuss have found their way into mainstream production systems serving millions of users on a daily basis.

This survey considers research published until January 2015. We refer to the final (journal) version of a result, citing conference publications only if a journal version is not yet available. The reader should keep in mind that the journal publications we cite often report on work that first appeared (at a conference) much earlier.

2 Shortest Paths Algorithms

Let $G = (V, A)$ be a (directed) graph with a set V of vertices and a set A of arcs. Each arc $(u, v) \in A$ has an associated nonnegative *length* $\ell(u, v)$. The length of a path is the sum of its arc lengths. In the *point-to-point shortest path problem*, one is given as input the graph G , a source $s \in V$, and a target $t \in V$, and must compute the length of the shortest path from s to t in G . This is also denoted as $\text{dist}(s, t)$, the distance between s and t . The *one-to-all* problem is to compute the distances from a given vertex s to all vertices of the graph. The *all-to-one* problem is to find the distances from all vertices to s . The *many-to-many* problem is as follows: given a set S of sources and a set T of targets, find the distances $\text{dist}(s, t)$ for all $s \in S, t \in T$. For $S = T = V$ we have the *all pairs shortest path* problem.

In addition to the distances, many applications need to find the corresponding shortest paths. An *out-shortest path tree* is a compact representation of one-to-all shortest paths from the root r . (Likewise, the in-shortest path tree represents the all-to-one paths.) For each vertex $u \in V$, the path from r to u in the tree is the shortest path.

In this section, we focus on the basic point-to-point shortest path problem under the basic *server model*. We assume that all data fits in RAM. However, locality matters, and algorithms with fewer cache misses run faster. For some algorithms, we consider multi-core and machine-tailored implementations. In our model, preprocessing may be performed on a more powerful machine than queries (e.g., a machine with more memory). While preprocessing may take a long time (e.g., hours), queries need to be fast enough for interactive applications.

In this section, we first discuss basic techniques, then those using preprocessing. Since all methods discussed could in principle be applied to arbitrary graphs, we keep the description as general as possible. For intuition, however, it pays to keep road networks in mind, considering that they were the motivating application for most approaches we consider. We will explicitly consider road networks, including precise performance numbers, in Sect. 3.

2.1 Basic Techniques

The standard solution to the one-to-all shortest path problem is Dijkstra's algorithm [108]. It maintains a priority queue Q of vertices ordered by (tentative) distances from s . The algorithm initializes all distances to infinity, except $\text{dist}(s, s) = 0$, and adds s to Q . In each iteration, it extracts a vertex u with minimum distance from Q and *scans* it, i.e., looks at all arcs $a = (u, v) \in A$ incident to u . For each such arc, it determines the distance to v via arc a by computing $\text{dist}(s, u) + \ell(a)$. If this value improves $\text{dist}(s, v)$, the algorithm performs an arc *relaxation*: it updates $\text{dist}(s, v)$ and adds vertex v with key $\text{dist}(s, v)$ to the priority queue Q . Dijkstra's algorithm has the *label-setting* property: once a vertex $u \in V$ is scanned (settled), its distance value $\text{dist}(s, u)$ is correct. Therefore, for point-to-point queries, the algorithm may stop as soon as it scans the

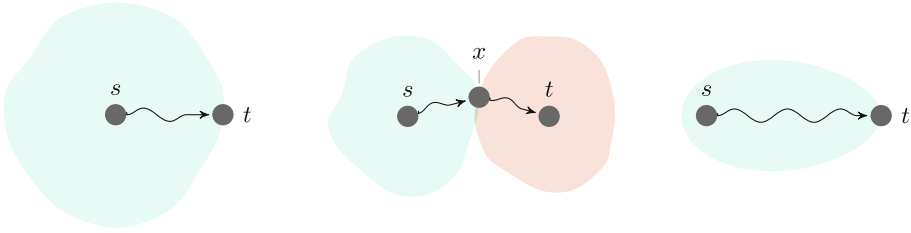


Fig. 1. Schematic search spaces of Dijkstra’s algorithm (left), bidirectional search (middle), and the A* algorithm (right).

target t . We refer to the set of vertices $S \subseteq V$ scanned by the algorithm as its *search space*. See Fig. 1 for an illustration.

The running time of Dijkstra’s algorithm depends on the priority queue used. The running time is $\mathcal{O}((|V| + |A|) \log |V|)$ with binary heaps [254], improving to $\mathcal{O}(|A| + |V| \log |V|)$ with Fibonacci heaps [129]. For arbitrary (non-integral) costs, generalized versions of binary heaps (such as 4-heaps or 8-heaps) tend to work best in practice [61]. If all arc costs are integers in the range $[0, C]$, multi-level buckets [103] yield a running time of $\mathcal{O}(|A| + |V| \sqrt{\log C})$ [8, 62] and work well in practice. For the average case, one can get an $\mathcal{O}(|V| + |A|)$ (linear) time bound [147, 192]. Thorup [244] has improved the theoretical worst-case bound of Dijkstra’s algorithm to $\mathcal{O}(|A| + |V| \log \log \min\{|V|, C\})$, but the required data structure is rather involved and unlikely to be faster in practice.

In practice, one can reduce the search space using *bidirectional search* [67], which simultaneously runs a forward search from s and a backward search from t . The algorithm may stop as soon as the intersection of their search spaces provably contains a vertex x on the shortest path from s to t . For road networks, bidirectional search visits roughly half as many vertices as the unidirectional approach.

An alternative method for computing shortest paths is the Bellman-Ford algorithm [46, 127, 198]. It uses no priority queue. Instead, it works in rounds, each scanning all vertices whose distance labels have improved. A simple FIFO queue can be used to keep track of vertices to scan next. It is a *label-correcting* algorithm, since each vertex may be scanned multiple times. Although it runs in $\mathcal{O}(|V| |A|)$ time in the worst case, it is often much faster, making it competitive with Dijkstra’s algorithm in some scenarios. In addition, it works on graphs with negative edge weights.

Finally, the Floyd-Warshall algorithm [126] computes distances between *all* pairs of vertices in $\Theta(|V|^3)$ time. For sufficiently dense graphs, this is faster than $|V|$ calls to Dijkstra’s algorithm.

2.2 Goal-Directed Techniques

Dijkstra’s algorithm scans all vertices with distances smaller than $\text{dist}(s, t)$. Goal-directed techniques, in contrast, aim to “guide” the search toward the target by

avoiding the scans of vertices that are not in the direction of t . They either exploit the (geometric) embedding of the network or properties of the graph itself, such as the structure of shortest path trees toward (compact) regions of the graph.

A Search.* A classic goal-directed shortest path algorithm is A* search [156]. It uses a potential function $\pi: V \rightarrow \mathbb{R}$ on the vertices, which is a *lower bound* on the distance $\text{dist}(u, t)$ from u to t . It then runs a modified version of Dijkstra’s algorithm in which the priority of a vertex u is set to $\text{dist}(s, u) + \pi(u)$. This causes vertices that are closer to the target t to be scanned earlier during the algorithm. See Fig. 1. In particular, if π were an *exact* lower bound ($\pi(u) = \text{dist}(u, t)$), *only* vertices along shortest s - t paths would be scanned. More vertices may be visited in general but, as long as the potential function is *feasible* (i.e., if $\ell(v, w) - \pi(v) + \pi(w) \geq 0$ for $(v, w) \in E$), an s - t query can stop with the correct answer as soon as it is about to scan the target vertex t .

The algorithm can be made bidirectional, but some care is required to ensure correctness. A standard approach is to ensure that the forward and backward potential functions are consistent. In particular, one can combine two arbitrary feasible functions π_f and π_r into consistent potentials by using $(\pi_f - \pi_r)/2$ for the forward search and $(\pi_r - \pi_f)/2$ for the backward search [163]. Another approach, which leads to similar results in practice, is to change the stopping criterion instead of making the two functions consistent [148, 166, 216, 220].

In road networks with travel time metric, one can use the geographical distance [217, 237] between u and t divided by the maximum travel speed (that occurs in the network) as the potential function. Unfortunately, the corresponding bounds are poor, and the performance gain is small or non-existent [148]. In practice, the algorithm can be accelerated using more aggressive bounds (for example, a smaller denominator), but correctness is no longer guaranteed. In practice, even when minimizing travel distances in road networks, A* with geographical distance bound performs poorly compared to other modern methods.

One can obtain much better lower bounds (and preserve correctness) with the *ALT* (*A**, *landmarks*, and *triangle inequality*) algorithm [148]. During a preprocessing phase, it picks a small set $L \subseteq V$ of *landmarks* and stores the distances between them and all vertices in the graph. During an s - t query, it uses triangle inequalities involving the landmarks to compute a valid lower bound on $\text{dist}(u, t)$ for any vertex u . More precisely, for any landmark l_i , both $\text{dist}(u, t) \geq \text{dist}(u, l_i) - \text{dist}(t, l_i)$ and $\text{dist}(u, t) \geq \text{dist}(l_i, t) - \text{dist}(l_i, u)$ hold. If several landmarks are available, one can take the maximum overall bound. See Fig. 2 for an illustration. The corresponding potential function is feasible [148].

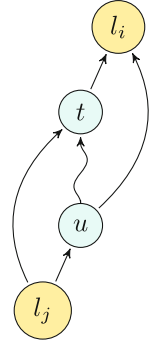


Fig. 2. Triangle inequalities for ALT.

The quality of the lower bounds (and thus query performance) depends on which vertices are chosen as landmarks during preprocessing. In road networks,

picking well-spaced landmarks close to the boundary of the graph leads to the best results, with acceptable query times on average [112, 150]. For a small (but noticeable) fraction of the queries, however, speedups relative to bidirectional Dijkstra are minor.

Geometric Containers. Another goal-directed method is *Geometric Containers*. It precomputes, for each arc $a = (u, v) \in A$, an arc label $L(a)$ that encodes the set V_a of vertices to which a shortest path from u begins with the arc a . Instead of storing V_a explicitly, $L(a)$ approximates this set by using geometric information (i. e., the coordinates) of the vertices in V_a . During a query, if the target vertex t is not in $L(a)$, the search can safely be pruned at a . Schulz et al. [235] approximate the set V_a by an angular sector (centered at u) that covers all vertices in V_a . Wagner et al. [251] consider other geometric containers, such as ellipses and the convex hull, and conclude that bounding boxes perform consistently well. For graphs with no geometric information, one can use graph layout algorithms and then create the containers [55, 250]. A disadvantage of Geometric Containers is that its preprocessing essentially requires an all-pairs shortest path computation, which is costly.

Arc Flags. The *Arc Flags* approach [157, 178] is somewhat similar to Geometric Containers, but does not use geometry. During preprocessing, it partitions the graph into K cells that are roughly *balanced* (have similar number of vertices) and have a small number of boundary vertices. Each arc maintains a vector of K bits (arc flags), where the i -th bit is set if the arc lies on a shortest path to some vertex of cell i . The search algorithm then prunes arcs which do not have the bit set for the cell containing t . For better query performance, arc flags can be extended to nested multilevel partitions [197]. Whenever the search reaches the cell that contains t , it starts evaluating arc flags with respect to the (finer) cells of the level below. This approach works best in combination with bidirectional search [157].

The arc flags for a cell i are computed by growing a backward shortest path tree from each boundary vertex (of cell i), setting the i -th flag for all arcs of the tree. Alternatively, one can compute arc flags by running a label-correcting algorithm from all boundary vertices simultaneously [157]. To reduce preprocessing space, one can use a compression scheme that flips some flags from zero to one [58], which preserves correctness. As Sect. 3 will show, Arc Flags currently have the fastest query times among purely goal-directed methods for road networks. Although high preprocessing times (of several hours) have long been a drawback of Arc Flags, the recent PHAST algorithm (cf. Sect. 2.7) can make this method more competitive with other techniques [75].

Precomputed Cluster Distances. Another goal-directed technique is *Precomputed Cluster Distances* (PCD) [188]. Like Arc Flags, it is based on a (preferably balanced) partition $\mathcal{C} = (C_1, \dots, C_K)$ with K cells (or clusters). The preprocessing algorithm computes the shortest path distances between all pairs of cells.

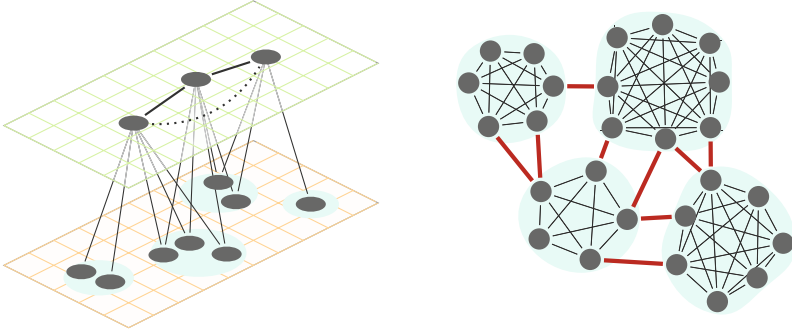


Fig. 3. Left: Multilevel overlay graph with two levels. The dots depict separator vertices in the lower and upper level. Right: Overlay graph constructed from arc separators. Each cell contains a full clique between its boundary vertices, and cut arcs are thicker.

The query algorithm is a pruned version of Dijkstra’s algorithm. For any vertex u visited by the search, a valid lower bound on its distance to the target is $\text{dist}(s, u) + \text{dist}(C(u), C(t)) + \text{dist}(v, t)$, where $C(u)$ is the cell containing u and v is the boundary vertex of $C(t)$ that is closest to t . If this bound exceeds the best current upper bound on $\text{dist}(s, t)$, the search is pruned. For road networks, PCD has similar query times to ALT, but requires less space.

Compressed Path Databases. The Compressed Path Databases (CPD) [52, 53] method implicitly stores all-pairs shortest path information so that shortest paths can be quickly retrieved during queries. Each vertex $u \in V$ maintains a label $L(u)$ that stores the *first move* (the arc incident to u) of the shortest path toward *every* other vertex v of the graph. A query from s simply scans $L(u)$ for t , finding the first arc (s, u) of the shortest path (to t); it then recurses on u until it reaches t . Explicitly storing the first arc of every shortest path (in $\Theta(|V|^2)$ space) would be prohibitive. Instead, Botea and Harabor [53] propose a lossless data compression scheme that groups vertices that share the same first move (out of u) into nonoverlapping geometric rectangles, which are then stored with u . Further optimizations include storing the most frequent first move as a default and using more sophisticated compression techniques. This leads to fast queries, but space consumption can be quite large; the method is thus dominated by other approaches. CPD can be seen as an evolution of the *Spatially Induced Linkage Cognizance* (SILC) algorithm [228], and both can be seen as stronger versions of Geometric Containers.

2.3 Separator-Based Techniques

Planar graphs have small (and efficiently-computable) separators [181]. Although road networks are not planar (think of tunnels or overpasses), they have been observed to have small separators as well [79, 123, 227]. This fact is exploited by the methods in this section.

Vertex Separators. We first consider algorithms based on *vertex separators*. A vertex separator is a (preferably small) subset $S \subset V$ of the vertices whose removal decomposes the graph G into several (preferably balanced) cells (components). This separator can be used to compute an *overlay graph* G' over S . Shortcut arcs [249] are added to the overlay such that distances between *any* pair of vertices from S are preserved, i.e., they are equivalent to the distance in G . The much smaller overlay graph can then be used to accelerate (parts of) the query algorithm.

Schulz et al. [235] use an overlay graph over a carefully chosen subset S (not necessarily a separator) of “important” vertices. For each pair of vertices $u, v \in S$, an arc (u, v) is added to the overlay if the shortest path from u to v in G does not contain any other vertex w from S . This approach can be further extended [160, 236] to multilevel hierarchies. In addition to arcs between separator vertices of the same level, the overlay contains, for each cell on level i , arcs between the confining level i separator vertices and the interior level $(i - 1)$ separator vertices. See Fig. 3 for an illustration.

Other variants of this approach offer different trade-offs by adding many more shortcuts to the graph during preprocessing, sometimes across different levels [151, 164]. In particular *High-Performance Multilevel Routing* (HPML) [83] substantially reduces query times but significantly increases the total space usage and preprocessing time. A similar approach, based on path separators for planar graphs, was proposed by Thorup [245] and implemented by Muller and Zachariasen [205]. It works reasonably well to find approximate shortest paths on undirected, planarized versions of road networks.

Arc Separators. The second class of algorithms we consider uses *arc separators* to build the overlay graphs. In a first step, one computes a partition $\mathcal{C} = (C_1, \dots, C_k)$ of the vertices into balanced cells while attempting to minimize the number of cut arcs (which connect boundary vertices of different cells). Shortcuts are then added to preserve the distances between the boundary vertices within each cell.

An early version of this approach is the *Hierarchical Multi* (HiTi) method [165]. It builds an overlay graph containing all boundary vertices and all cut arcs. In addition, for each pair u, v of boundary vertices in C_i , HiTi adds to the overlay a shortcut (u, v) representing the shortest path from u to v in G restricted to C_i . The query algorithm then (implicitly) runs Dijkstra’s algorithm on the subgraph induced by the cells containing s and t plus the overlay. This approach can be extended to use nested multilevel partitions. HiTi has only been tested on grid graphs [165], leading to modest speedups. See also Fig. 3.

The recent *Customizable Route Planning* (CRP) [76, 78] algorithm uses a similar approach, but is specifically engineered to meet the requirements of real-world systems operating on road networks. In particular, it can handle turn costs and is optimized for fast updates of the cost function (metric). Moreover, it uses PUNCH [79], a graph partitioning algorithm tailored to road networks. Finally, CRP splits preprocessing in two phases: metric-independent preprocessing and customization. The first phase computes, besides the multilevel partition, the

topology of the overlays, which are represented as matrices in contiguous memory for efficiency. Note that the partition does not depend on the cost function. The second phase (which takes the cost function as input) computes the costs of the clique arcs by processing the cells in bottom-up fashion and in parallel. To process a cell, it suffices to run Dijkstra’s algorithm from each boundary vertex, but the second phase is even faster using the Bellman-Ford algorithm paired with (metric-independent) contraction [100] (cf. Sect. 2.4), at the cost of increased space usage. Further acceleration is possible using GPUs [87]. Queries are bidirectional searches in the overlay graph, as in HiTi.

2.4 Hierarchical Techniques

Hierarchical methods aim to exploit the inherent hierarchy of road networks. Sufficiently long shortest paths eventually converge to a small arterial network of important roads, such as highways. Intuitively, once the query algorithm is far from the source and target, it suffices to only scan vertices of this subnetwork. In fact, using input-defined road categories in this way is a popular heuristic [115, 158], though there is no guarantee that it will find exact shortest paths. Fu et al. [130] give an overview of early approaches using this technique. Since the algorithms we discuss must find exact shortest paths, their correctness must not rely on unverifiable properties such as input classifications. Instead, they use the preprocessing phase to compute the importance of vertices or arcs according to the actual shortest path structure.

Contraction Hierarchies. An important approach to exploiting the hierarchy is to use *shortcuts*. Intuitively, one would like to augment G with shortcuts that could be used by long-distance queries to skip over “unimportant” vertices.

The *Contraction Hierarchies* (CH) algorithm, proposed by Geisberger et al. [142], implements this idea by repeatedly executing a *vertex contraction* operation. To contract a vertex v , it is (temporarily) removed from G , and a shortcut is created between each pair u, w of neighboring vertices if the shortest path from u to w is unique and contains v . During preprocessing, CH (heuristically) orders the vertices by “importance” and contracts them from least to most important.

The query stage runs a bidirectional search from s and t on G augmented by the shortcuts computed during preprocessing, but only visits arcs leading to vertices of higher ranks (importance). See Fig. 4 for an illustration. Let $d_s(u)$ and $d_t(u)$ be the corresponding distance labels obtained by these *upward* searches (set to ∞ for vertices that are not visited). It is easy to show that $d_s(u) \geq \text{dist}(s, u)$ and $d_t(u) \geq \text{dist}(u, t)$; equality is not guaranteed due to

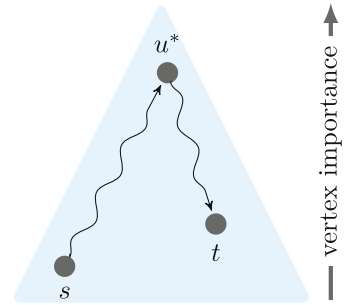


Fig. 4. Illustrating a Contraction Hierarchies query.

pruning. Nevertheless, Geisberger et al. [142] prove that the highest-ranked vertex u^* on the original s - t path will be visited by both searches, and that both its labels will be exact, i. e., $d_s(u^*) = \text{dist}(s, u^*)$ and $d_t(u^*) = \text{dist}(u^*, t)$. Therefore, among all vertices u visited by both searches, the one minimizing $d_s(u) + d_t(u)$ represents the shortest path. Note that, since u^* is not necessarily the first vertex that is scanned by both searches, they cannot stop as soon as they meet.

Query times depend on the vertex order. During preprocessing, the vertex order is usually determined online and bottom-up. The overall (heuristic) goal is to minimize the number of edges added during preprocessing. One typically selects the vertex to be contracted next by considering a combination of several factors, including the net number of shortcuts added and the number of nearby vertices already contracted [142, 168]. Better vertex orders can be obtained by combining the bottom-up algorithm with (more expensive) top-down offline algorithms that explicitly classify vertices hitting many shortest paths as more important [5, 77]. Since road networks have very small separators [79], one can use nested dissection to obtain reasonably good orders that work for any length function [100, 107]. Approximate CH has been considered as a way to accommodate networks with less inherent hierarchy [143].

CH is actually a successor of Highway Hierarchies [225] and Highway Node Routing [234], which are based on similar ideas. CH is not only faster, but also conceptually simpler. This simplicity has made it quite versatile, serving as a building block not only for other point-to-point algorithms [4, 15, 40, 100], but also for extended queries (cf. Sect. 2.7) and applications (cf. Sect. 3.2).

Reach. An earlier hierarchical approach is *Reach* [154]. Reach is a centrality measure on vertices. Let P be a shortest s - t path that contains vertex u . The reach $r(u, P)$ of u with respect to P is defined as $\min\{\text{dist}(s, u), \text{dist}(u, t)\}$. The (global) reach of u in the graph G is the maximum reach of u over all shortest paths that contain u . Like other centrality measures [54], reach captures the importance of vertices in the graph, with the advantage that it can be used to prune a Dijkstra-based search.

A reach-based s - t query runs Dijkstra's algorithm, but prunes the search at any vertex u for which both $\text{dist}(s, u) > r(u)$ and $\text{dist}(u, t) > r(u)$ hold; the shortest s - t path provably does not contain u . To check these conditions, it suffices [149] to run bidirectional searches, each using the radius of the opposite search as a lower bound on $\text{dist}(u, t)$ (during the forward search) or $\text{dist}(s, u)$ (backward search).

Reach values are determined during the preprocessing stage. Computing exact reaches requires computing shortest paths for all pairs of vertices, which is too expensive on large road networks. But the query is still correct if $r(u)$ represents only an upper bound on the reach of u . Gutman [154] has shown that such bounds can be obtained faster by computing partial shortest path trees. Goldberg et al. [149] have shown that adding shortcuts to the graph effectively reduces the reaches of most vertices, drastically speeding up both queries and preprocessing and making the algorithm practical for continent-sized networks.

2.5 Bounded-Hop Techniques

The idea behind bounded-hop techniques is to precompute distances between pairs of vertices, implicitly adding “virtual shortcuts” to the graph. Queries can then return the length of a virtual path with very few hops. Furthermore, they use only the precomputed distances between pairs of vertices, and not the input graph. A naïve approach is to use single-hop paths, i. e., precompute the distances among *all* pairs of vertices $u, v \in V$. A single table lookup then suffices to retrieve the shortest distance. While the recent PHAST algorithm [75] has made precomputing all-pairs shortest paths feasible, storing all $\Theta(|V|^2)$ distances is prohibitive already for medium-sized road networks. As we will see in this section, considering paths with slightly more hops (two or three) leads to algorithms with much more reasonable trade-offs.

Labeling Algorithms. We first consider *labeling algorithms* [215]. During preprocessing, a *label* $L(u)$ is computed for each vertex u of the graph, such that, for any pair u, v of vertices, the distance $\text{dist}(u, v)$ can be determined by only looking at the labels $L(u)$ and $L(v)$. A natural special case of this approach is *Hub Labeling* (HL) [64, 135], in which the label $L(u)$ associated with vertex u consists of a set of vertices (the *hubs* of u), together with their distances from u . These labels are chosen such that they obey the *cover property*: for any pair (s, t) of vertices, $L(s) \cap L(t)$ must contain at least one vertex on the shortest s - t path. Then, the distance $\text{dist}(s, t)$ can be determined in linear (in the label size) time by evaluating $\text{dist}(s, t) = \min\{\text{dist}(s, u) + \text{dist}(u, t) \mid u \in L(s) \text{ and } u \in L(t)\}$. See Fig. 5 for an illustration. For directed graphs, the label associated with u is actually split in two: the forward label $L_f(u)$ has distances from u to the hubs, while the backward label $L_b(u)$ has distances from the hubs to u ; the shortest s - t path has a hub in $L_f(s) \cap L_b(t)$.

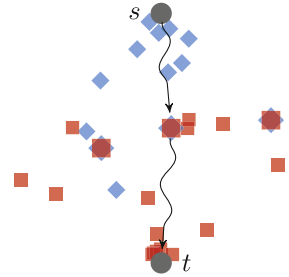


Fig. 5. Illustrating hub labels of vertices s (diamonds) and t (squares).

Although the required average label size can be $\Theta(|V|)$ in general [135], it can be significantly smaller for some graph classes. For road networks, Abraham et al. [4] have shown that one can obtain good results by defining the label of vertex u as the (upward) search space of a CH query from u (with suboptimal entries removed). In general, any vertex ordering fully defines a labeling [5], and an ordering can be converted into the corresponding labeling efficiently [5, 12]. The CH-induced order works well for road networks. For even smaller labels, one can pick the most important vertices greedily, based on how many shortest paths they hit [5]. A sampling version of this greedy approach works efficiently for a wide range of graph classes [77].

Note that, if labels are sorted by hub ID, a query consists of a linear sweep over two arrays, as in mergesort. Not only is this approach very simple, but it also has an almost perfect locality of access. With careful engineering, one

does not even have to look at all the hubs in a label [4]. As a result, HL has the fastest known queries for road networks, taking roughly the time needed for five accesses to main memory (see Sect. 3.1). One drawback is space usage, which, although not prohibitive, is significantly higher than for competing methods. By combining common substructures that appear in multiple labels, *Hub Label Compression* (HLC) [82] (see also [77]) reduces space usage by an order of magnitude, at the expense of higher query times.

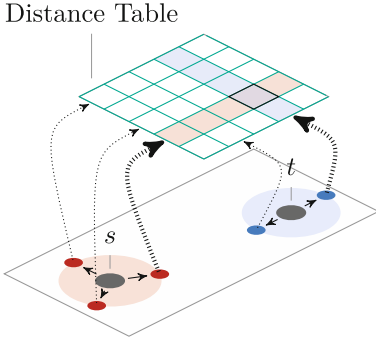


Fig. 6. Illustrating a TNR query. The access nodes of s (t) are indicated by three (two) dots. The arrows point to the respective rows/columns of the distance table. The highlighted entries correspond to the access nodes which minimize the combined s - t distance.

To account for such cases, a *locality filter* decides whether the query might be local (i.e., does not contain a vertex from T). In that case, a fallback shortest path algorithm (typically CH) is run to compute the correct distance. Note that TNR is still correct even if the locality filter occasionally misclassifies a global query as local. See Fig. 6 for an illustration of a TNR query. Interestingly, global TNR queries (which use the distance tables) tend to be faster than local ones (which perform graph searches). To accelerate local queries, TNR can be extended to multiple (hierarchical) layers of transit (and access) nodes [28, 224].

The choice of the transit node set is crucial to the performance of the algorithm. A natural approach is to select vertex separators or boundary vertices of arc separators as transit nodes. In particular, using grid-based separators yields natural locality filters and works well enough in practice for road networks [28]. (Although an optimized preprocessing routine for this grid-based approach was later shown to have a flaw that could potentially result in sub-optimal queries [257], the version with slower preprocessing reported in [28] is correct and achieves the same query times.)

For better performance [3, 15, 142, 224], one can pick as transit nodes vertices that are classified as important by a hierarchical speedup technique (such as CH).

Transit Node Routing. The *Transit Node Routing* (TNR) [15, 28, 30, 224] technique uses distance tables on a subset of the vertices. During preprocessing, it selects a small set $T \subseteq V$ of *transit nodes* and computes all pairwise distances between them. From those, it computes, for each vertex $u \in V \setminus T$, a relevant set of *access nodes* $A(u) \subseteq T$. A transit node $v \in T$ is an access node of u if there is a shortest path P from u in G such that v is the first transit node contained in P . In addition to the vertex itself, preprocessing also stores the distances between u and its access nodes.

An s - t query uses the distance table to select the path that minimizes the combined s - $a(s)$ - $a(t)$ - t distance, where $a(s) \in A(s)$ and $a(t) \in A(t)$ are access nodes. Note that the result is incorrect if the shortest path does not contain a vertex from T .

Locality filters are less straightforward in such cases: although one can still use geographical distances [142, 224], a graph-based approach considering the Voronoi regions [189] induced by transit nodes tends to be significantly more accurate [15]. A theoretically justified TNR variant [3] also picks important vertices as transit nodes and has a natural graph-based locality filter, but is impractical for large networks.

Pruned Highway Labeling. The Pruned Highway Labeling (PHL) [11] algorithm can be seen as a hybrid between pure labeling and transit nodes. Its preprocessing routine decomposes the input into disjoint shortest paths, then computes a label for each vertex v containing the distance from v to vertices in a small subset of such paths. The labels are such that any shortest s - t path can be expressed as s - u - w - t , where u - w is a subpath of a path P that belongs to the labels of s and t . Queries are thus similar to HL, finding the lowest-cost intersecting path. For efficient preprocessing, the algorithm uses the pruned labeling technique [12]. Although this method has some similarity with Thorup’s distance oracle for planar graphs [245], it does not require planarity. PHL has only been evaluated on undirected graphs, however.

2.6 Combinations

Since the individual techniques described so far exploit different graph properties, they can often be combined for additional speedups. This section describes such hybrid algorithms. In particular, early results [161, 235] considered the combination of Geometric Containers, multilevel overlay graphs, and (Euclidean-based) A* on transportation networks, resulting in speedups of one or two orders of magnitude over Dijkstra’s algorithm.

More recent studies have focused on combining hierarchical methods (such as CH or Reach) with fast goal-directed techniques (such as ALT or Arc Flags). For instance, the *REAL* algorithm combines Reach and ALT [149]. A basic combination is straightforward: one simply runs an ALT query with additional pruning by reach (using the ALT lower bounds themselves for reach evaluations). A more sophisticated variant uses *reach-aware landmarks*: landmarks and their distances are only precomputed for vertices with high reach values. This saves space (only a small fraction of the graph needs to store landmark distances), but requires two-stage queries (goal direction is only used when the search is far enough from both source and target).

A similar space-saving approach is used by *Core-ALT* [40, 88]. It first computes an overlay graph for the *core graph*, a (small) subset (e. g., 1 %) of vertices (which remain after “unimportant” ones are contracted), then computes landmarks for the core vertices only. Queries then work in two stages: first plain bidirectional search, then ALT is applied when the search is restricted to the core. The (earlier) *HH** approach [95] is similar, but uses Highway Hierarchies [225] to determine the core.

Another approach with two-phase queries is *ReachFlags* [40]. During preprocessing, it first computes (approximate) reach values for all vertices in G ,

then extracts the subgraph H induced by all vertices whose reach value exceeds a certain threshold. Arc flags are then only computed for H , to be used in the second phase of the query.

The *SHARC* algorithm [39] combines the computation of shortcuts with multilevel arc flags. The preprocessing algorithm first determines a partition of the graph and then computes shortcuts and arc flags in turn. Shortcuts are obtained by contracting unimportant vertices with the restriction that shortcuts never span different cells of the partition. The algorithm then computes arc flags such that, for each cell C , the query uses a shortcut arc if and only if the target vertex is not in C . Space usage can be reduced with various compression techniques [58]. Note that *SHARC* is unidirectional and hierarchical: arc flags not only guide the search toward the target, but also vertically across the hierarchy. This is useful when the backward search is not well defined, as in time-dependent route planning (discussed in Sect. 2.7).

Combining CH with Arc Flags results in the *CHASE* algorithm [40]. During preprocessing, a regular contraction hierarchy is computed and the search graph that includes all shortcuts is assembled. The algorithm then extracts the subgraph H induced by the top k vertices according to the contraction order. Bidirectional arc flags (and the partition) are finally computed on the restricted subgraph H . Queries then run in two phases. Since computing arc flags was somewhat slow, k was originally set to a small fraction (about 5%) of the total number $|V|$ of vertices [40]. More recently, Dellinger et al. showed that *PHAST* (see Sect. 2.7) can compute arc flags fast enough to allow k to be set to $|V|$, making *CHASE* queries much simpler (single-pass), as well as faster [75].

Finally, Bauer et al. [40] combine Transit Node Routing with Arc Flags to obtain the TNR+AF algorithm. Recall that the bottleneck of the TNR query is performing the table lookups between pairs of access nodes from $A(s)$ and $A(t)$. To reduce the number of lookups, TNR+AF's preprocessing decomposes the set of transit nodes T into k cells. For each vertex s and access node $u \in A(s)$, it stores a k -bit vector, with bit i indicating whether there exists a shortest path from s to cell i through u . A query then only considers the access nodes from s that have their bits set with respect to the cells of $A(t)$. A similar pruning is done at the target.

2.7 Extensions

In various applications, one is often interested in more than just the length of the shortest path between two points in a static network. Most importantly, one should also be able to retrieve the shortest path itself. Moreover, many of the techniques considered so far can be adapted to compute batched shortest paths (such as distance tables), to more realistic scenarios (such as dynamic networks), or to deal with multiple objective functions. In the following, we briefly discuss each of these extensions.

Path Retrieval. Our descriptions so far have focused on finding only the *length* of the shortest path. The algorithms we described can easily be augmented to

provide the actual list of edges or vertices on the path. For techniques that do not use shortcuts (such as Dijkstra’s algorithm, A* search, or Arc Flags), one can simply maintain a parent pointer for each vertex v , updating it whenever the distance label of v changes. When shortcuts are present (such as in CH, SHARC, or CRP), this approach gives only a *compact* representation of the shortest path (in terms of shortcuts). The shortcuts then need to be unpacked. If each shortcut is the concatenation of two other arcs (or shortcuts), as in CH, storing the middle vertex [142] of each shortcut allows for an efficient (linear-time) recursive unpacking of all shortcuts on the output path. If shortcuts are built from multiple arcs (as for CRP or SHARC), one can either store the entire sequence for each shortcut [225] or run a local (bidirectional) Dijkstra search from its endpoints [78]. These two techniques can be used for bounded-hop algorithms as well.

Batched Shortest Paths. Some applications require computing multiple paths at once. For example, advanced logistics applications may need to compute all distances between a source set S and a target set T . This can be trivially done with $|S| \cdot |T|$ point-to-point shortest-path computations. Using a hierarchical speedup technique (such as CH), this can be done in time comparable to $\mathcal{O}(|S| + |T|)$ point-to-point queries in practice, which is much faster. First, one runs a backward upward search from each $t_i \in T$; for each vertex u scanned during the search from t_i , one stores its distance label $d_{t_i}(u)$ in a bucket $\beta(u)$. Then, one runs a forward upward search from each $s_j \in S$. Whenever such a search scans a vertex v with a non-empty bucket, one searches the bucket and checks whether $d_{s_j}(v) + d_{t_i}(v)$ improves the best distance seen so far between s_j and t_i . This *bucket-based approach* was introduced for Highway Hierarchies [172], but can be used with any other hierarchical speedup technique (such as CH) and even with hub labels [81]. When the bucket-based approach is combined with a separator-based technique (such as CRP), it is enough to keep buckets only for the boundary vertices [99]. Note that this approach can be used to compute one-to-many or many-to-many distances.

Some applications require one-to-all computations, i. e., finding the distances from a source vertex s to all other vertices in the graph. For this problem, Dijkstra’s algorithm is optimal in the sense that it visits each edge exactly once, and hence runs in essentially linear time [147]. However, Dijkstra’s algorithm has bad locality and is hard to parallelize, especially for sparse graphs [186, 193]. PHAST [75] builds on CH to improve this. The idea is to split the search in two phases. The first is a forward upward search from s , and the second runs a linear scan over the shortcut-enriched graph, with distance values propagated from more to less important vertices. Since the instruction flow of the second phase is (almost) independent of the source, it can be engineered to exploit parallelism and improve locality. In road networks, PHAST can be more than an order of magnitude faster than Dijkstra’s algorithm, even if run sequentially, and can be further accelerated using multiple cores and even GPUs. This approach can also be extended to the *one-to-many problem*, i. e., computing distances from

a source to a subset of predefined targets [81]. Similar techniques can also be applied with graph separators (instead of CH), yielding comparable query times but with faster (metric-dependent) preprocessing [113].

Dynamic Networks. Transportation networks tend to be dynamic, with unpredictable delays, traffic, or closures. If one assumes that the modified network is stable for the foreseeable future, the obvious approach for speedup techniques to deal with this is to rerun the preprocessing algorithm. Although this ensures queries are as fast as in the static scenario, it can be quite costly. As a result, four other approaches have been considered.

It is often possible to just “repair” the preprocessed data instead of rebuilding it from scratch. This approach has been tried for various techniques, including Geometric Containers [251], ALT [96], Arc Flags [66], and CH [142, 234], with varying degrees of success. For CH, for example, one must keep track of dependencies between shortcuts, partially rerunning the contraction as needed. Changes that affect less important vertices can be dealt with faster.

Another approach is to adapt the query algorithms to work around the “wrong” parts of the preprocessing phase. In particular, ALT is resilient to increases in arc costs (due to traffic, for example): queries remain correct with the original preprocessing, though query times may increase [96]. Less trivially, CH queries can also be modified to deal with dynamic changes to the network [142, 234] by allowing the search to bypass affected shortcuts by going “down” the hierarchy. This is useful when queries are infrequent relative to updates.

A third approach is to make the preprocessing stage completely metric-independent, shifting all metric-dependent work to the query phase. Funke et al. [131] generalize the multilevel overlay graph approach to encode *all* k -hop paths (for small k) in an overlay graph. Under the assumption that edge costs are defined by a small number of physical parameters (as in simplified road networks) this allows setting the edge costs at query time, though queries become significantly slower.

For more practical queries, the fourth approach splits the preprocessing phase into metric-independent and metric-dependent stages. The metric-independent phase takes as input only the network topology, which is fairly stable. When edge costs change (which happens often), only the (much cheaper) metric-dependent stage must be rerun, partially or in full. This concept can again be used for various techniques, with ALT, CH, and CRP being the most prominent. For ALT, one can keep the landmarks, and just recompute the distances to them [96, 112]. For CH, one can keep the ordering, and just rerun contraction [107, 142]. For CRP, one can keep the partitioning and the overlay topology, and just recompute the shortcut lengths using a combination of contraction and graph searches [78]. Since the contraction is metric-independent, one can precompute and store the sequence of contraction operations and reexecute them efficiently whenever edge lengths change [78, 87]. The same approach can be used for CH with metric-independent orders [107].

Time-Dependence. In real transportation networks, the best route often depends on the departure time in a predictable way [102]. For example, certain roads are consistently congested during rush hours, and certain buses or trains run with different frequencies during the day. When one is interested in the earliest possible arrival given a specified departure time (or, symmetrically, the latest departure), one can model this as the *time-dependent* shortest path problem, which assigns travel time functions to (some of) the edges, representing how long it takes to traverse them at each time of the day. Dijkstra’s algorithm still works [65] as long as later departures cannot lead to earlier arrivals; this *non-overtaking* property is often called first-in first-out (FIFO). Although one must deal with functions instead of scalars, the theoretical running time of Dijkstra-based algorithms can still be bounded [71, 128]. Moreover, many of the techniques described so far work in this scenario, including bidirectional ALT [88, 207], CH [32], or SHARC [72]. Recently, Kontogiannis and Zaroliagis [175] have introduced a theoretical (approximate) distance oracle with sublinear running time. Other scenarios (besides FIFO with no waiting at vertices) have been studied [69, 70, 208, 209], but they are less relevant for transportation networks.

There are some challenges, however. In particular, bidirectional search becomes more complicated (since the time of arrival is not known), requiring changes to the backward search [32, 207]. Another challenge is that shortcuts become more space-consuming (they must model a more complicated travel time function), motivating compression techniques that do not sacrifice correctness, as demonstrated for SHARC [58] or CH [32]. Batched shortest paths can be computed in such networks efficiently as well [141].

Time-dependent networks motivate some elaborate (but still natural) queries, such as finding the best departure time in order to minimize the total time in transit. Such queries can be dealt with by *range searches*, which compute the travel time function between two points. There exist Dijkstra-based algorithms [71] for this problem, and most speedup techniques can be adapted to deal with this as well [32, 72].

Unfortunately, even a slight deviation from the travel time model, where total cost is a linear combination of travel time and a constant cost offset, makes the problem NP-hard [9, 33]. However, a heuristic adaptation of time-dependent CH shows negligible errors in practice [33].

Multiple Objective Functions. Another natural extension is to consider multiple cost functions. For example, certain vehicle types cannot use all segments of the transportation network. One can either adapt the preprocessing such that these *edge restrictions* can be applied during query time [140], or perform a metric update for each vehicle type.

Also, the search request can be more flexible. For example, one may be willing to take a more scenic route even if the trip is slightly longer. This can be dealt with by performing a multicriteria search. In such a search, two paths are incomparable if neither is better than the other in all criteria. The goal is

to find a *Pareto set*, i.e., a maximum set of incomparable paths. Such sets of shortest paths can be computed by extensions of Dijkstra’s algorithm; see [117] for a survey on multicriteria combinatorial optimization. More specifically, the *Multicriteria Label-Setting* (MLS) algorithm [155, 187, 196, 243] extends Dijkstra’s algorithm by keeping, for each vertex, a *bag* of nondominated labels. Each label is represented as a tuple, with one entry per optimization criterion. The priority queue maintains labels instead of vertices, typically ordered lexicographically. In each iteration, it extracts the minimum label L and scans the incident arcs $a = (u, v)$ of the vertex u associated with L . It does so by adding the cost of a to L and then merging L into the bag of v , eliminating possibly dominated labels on the fly. In contrast, the *Multi-Label-Correcting* (MLC) algorithm [68, 98] considers the whole bag of nondominated labels associated with u at once when scanning the vertex u . Hence, individual labels of u may be scanned multiple times during one execution of the algorithm.

Both MLS and MLC are fast enough as long as the Pareto sets are small [109, 204]. Unfortunately, Pareto sets may contain exponentially many solutions, even for the restricted case of two optimization criteria [155], which makes it hard to achieve large speedups [47, 97]. To reduce the size of Pareto sets, one can relax domination. In particular, $(1 + \varepsilon)$ -Pareto sets have provable polynomial size [212] and can be computed efficiently [182, 246, 253]. Moreover, large Pareto sets open up a potential for parallelization that is not present for a single objective function [124, 222].

A reasonable alternative [138] to multicriteria search is to optimize a linear combination $\alpha c_1 + (1 - \alpha)c_2$ of two criteria (c_1, c_2) , with the parameter α set at query time. Moreover, it is possible to efficiently compute the values of α where the path actually changes. Funke and Storandt [133] show that CH can handle such functions with polynomial preprocessing effort, even with more than two criteria.

2.8 Theoretical Results

Most of the algorithms mentioned so far were developed with practical performance in mind. Almost all methods we surveyed are exact: they provably find the exact shortest path. Their performance (in terms of both preprocessing and queries), however, varies significantly with the input graph. Most algorithms work well for real road networks, but are hardly faster than Dijkstra’s algorithm on some other graph classes. This section discusses theoretical work that helps understand why the algorithms perform well and what their limitations are.

Most of the algorithms considered have some degree of freedom during preprocessing (such as which partition, which vertex order, or which landmarks to choose). An obvious question is whether one could efficiently determine the best such choices for a particular input so as to minimize the query search space (a natural proxy for query times). Bauer et al. [36] have determined that finding optimal landmarks for ALT is NP-hard. The same holds for Arc Flags (with respect to the partition), SHARC (with respect to the shortcuts), Multilevel Overlay Graphs (with respect to the separator), Contraction Hierarchies (with

respect to the vertex order), and Hub Labels (with respect to the hubs) [252]. In fact, minimizing the number of shortcuts for CH is APX-hard [36, 194]. For SHARC, however, a greedy factor- k approximation algorithm exists [38]. Deciding which k shortcuts (for fixed k) to add to a graph in order to minimize the SHARC search space is also NP-hard [38]. Bauer et al. [35] also analyze the preprocessing of Arc Flags in more detail and on restricted graph classes, such as paths, trees, and cycles, and show that finding an optimal partition is NP-hard even for binary trees.

Besides complexity, theoretical performance bounds for query algorithms, which aim to explain their excellent practical performance, have also been considered. Proving better running time bounds than those of Dijkstra’s algorithm is unlikely for general graphs; in fact, there are inputs for which most algorithms are ineffective. That said, one can prove nontrivial bounds for specific graph classes. In particular, various authors [37, 194] have independently observed a natural relationship between CH and the notions of filled graphs [214] and elimination trees [232]. For planar graphs, one can use nested dissection [180] to build a CH order leading to $\mathcal{O}(|V| \log |V|)$ shortcuts [37, 194]. More generally, for minor-closed graph classes with balanced $\mathcal{O}(\sqrt{|V|})$ -separators, the search space is bounded by $\mathcal{O}(\sqrt{|V|})$ [37]. Similarly, on graphs with treewidth k , the search space of CH is bounded by $\mathcal{O}(k \log |V|)$ [37].

Road networks have motivated a large amount of theoretical work on algorithms for planar graphs. In particular, it is known that planar graphs have separators of size $\mathcal{O}(\sqrt{|V|})$ [180, 181]. Although road networks are not strictly planar, they do have small separators [79, 123], so theoretically efficient algorithms for planar graphs are likely to also perform well in road networks. Sommer [238] surveys several approximate methods with various trade-offs. In practice, the observed performance of most speedup techniques is much better on actual road networks than on arbitrary planar graphs (even grids). A theoretical explanation of this discrepancy thus requires a formalization of some property related to key features of real road networks.

One such graph property is *Highway Dimension*, proposed by Abraham et al. [3] (see also [1, 7]). Roughly speaking, a graph has highway dimension h if, at any scale r , one can hit all shortest paths of length at least r by a hitting set S that is *locally sparse*, in the sense that any ball of radius r has at most h elements from S . Based on previous experimental observations [30], the authors [7] conjecture that road networks have small highway dimension. Based on this notion, they establish bounds on the performance of (theoretically justified versions of) various speedup techniques in terms of h and the graph diameter D , assuming the graph is undirected and that edge lengths are integral. More precisely, after running a polynomial-time preprocessing routine, which adds $\mathcal{O}(h \log h \log D)$ shortcuts to G , Reach and CH run in $\mathcal{O}((h \log h \log D)^2)$ time. Moreover, they also show that HL runs in $\mathcal{O}(h \log h \log D)$ time and long-range TNR queries take $\mathcal{O}(h^2)$ time. In addition, Abraham et al. [3] show that a graph with highway dimension h has doubling dimension $\log(h + 1)$, and Kleinberg et al. [171] show that landmark-based triangulation yields good bounds for most pairs of

vertices of graphs with small doubling dimension. This gives insight into the good performance of ALT in road networks.

The notion of highway dimension is an interesting application of the scientific method. It was originally used to explain the good observed performance of CH, Reach, and TNR, and ended up predicting that HL (which had not been implemented yet) would have good performance in practice.

Generative models for road networks have also been proposed and analyzed. Abraham et al. [3, 7] propose a model that captures some of the properties of road networks and generates graphs with provably small highway dimension. Bauer et al. [42] show experimentally that several speedup techniques are indeed effective on graphs generated according to this model, as well as according to a new model based on Voronoi diagrams. Models with a more geometric flavor have been proposed by Eppstein and Goodrich [123] and by Eisenstat [118].

Besides these results, Rice and Tsotras [220] analyze the A* algorithm and obtain bounds on the search space size that depend on the underestimation error of the potential function. Also, maintaining and updating multilevel overlay graphs have been theoretically analyzed in [57]. For Transit Node Routing, Eisner and Funke [120] propose instance-based lower bounds on the size of the transit node set. For labeling algorithms, bounds on the label size for different graph classes are given by Gavaille et al. [135]. Approximation algorithms to compute small labels have also been studied [16, 64, 80]; although they can find slightly better labels than faster heuristics [5, 77], their running time is prohibitive [80].

Because the focus of this work is on algorithm engineering, we refrain from going into more detail about the available theoretical work. Instead, we refer the interested reader to overview articles with a more theoretical emphasis, such as those by Sommer [238], Zwick [262], and Gavaille and Peleg [134].

3 Route Planning in Road Networks

In this section, we experimentally evaluate how the techniques discussed so far perform in road networks. Moreover, we discuss applications of some of the techniques, as well as alternative settings such as databases or mobile devices.

3.1 Experimental Results

Our experimental analysis considers carefully engineered implementations, which is very important when comparing running times. They are written in C++ with custom-built data structures. Graphs are represented as adjacency arrays [190], and priority queues are typically binary heaps, 4-heaps, or multilevel buckets. As most arcs in road networks are bidirectional, state-of-the-art implementations use edge compression [233]: each road segment is stored at both of its endpoints, and each occurrence has two flags indicating whether the segment should be considered as an incoming and/or outgoing arc. This representation is compact and allows efficient iterations over incoming and outgoing arcs.

We give data for two models. The *simplified model* ignores turn restrictions and penalties, while the *realistic model* includes the turn information [255]. There are two common approaches to deal with turns. The *arc-based representation* [59] blows up the graph so that roads become vertices and feasible turns become arcs. In contrast, the *compact representation* [76, 144] keeps intersections as vertices, but with associated *turn tables*. One can save space by sharing turn tables among many vertices, since the number of intersection types in a road network is rather limited. Most speedup techniques can be used as is for the arc-based representation, but may need modification to work on the compact model.

Most experimental studies are restricted to the simplified model. Since some algorithms are more sensitive to how turns are modeled than others, it is hard to extrapolate these results to more realistic networks. We therefore consider experimental results for each model separately.

Simplified Model. An important driving force behind the research on speedup techniques for Dijkstra’s algorithm was its application to road networks. A key aspect for the success of this research effort was the availability of continent-sized benchmark instances. The most widely used instance has been the road network of Western Europe from PTV AG, with 18.0 million vertices and 42.5 million directed arcs. Besides ferries (for which the traversal time was given), it has 13 road categories. Category i has been assigned an average speed of $10i$ km/h. This synthetic assignment is consistent with more realistic proprietary data [78, 82]. Another popular (and slightly bigger) instance, representing the TIGER/USA road network, is undirected and misses several important road segments [6]. Although the inputs use the simplified model, they allowed researchers from various groups to run their algorithms on the same instance, comparing their performance. In particular, both instances were tested during the DIMACS Challenge on Shortest Paths [101].

Figure 7 succinctly represents the performance of previously published implementations of various point-to-point algorithms on the Western Europe instance, using travel time as the cost function. For each method, the plot relates its preprocessing and average query times. Queries compute the length of the shortest path (but not its actual list of edges) between sources and targets picked uniformly at random from the full graph. For readability, space consumption (a third important quality measure) is not explicitly represented.¹ We reproduce the numbers reported by Bauer et al. [40] for Reach, HH, HNR, ALT, (bidirectional) Arc Flags, REAL, HH*, SHARC, CALT, CHASE, ReachFlags and TNR+AF. For CHASE and Arc Flags, we also consider variants with quicker PHAST-based preprocessing [75]. In addition, we consider the recent ALT implementation by Efentakis and Pfoser [112]. Moreover, we report results for several variants of TNR [15, 40], Hub Labels [5, 82], HPML [83], Contraction Hierarchies (CH) [142], and Customizable Contraction Hierarchies (CCH) [107]. CRP (and the corresponding PUNCH) figures [78] use a more realistic graph

¹ The reader is referred to Sommer [238] for a similar plot (which inspired ours) relating query times to preprocessing space.

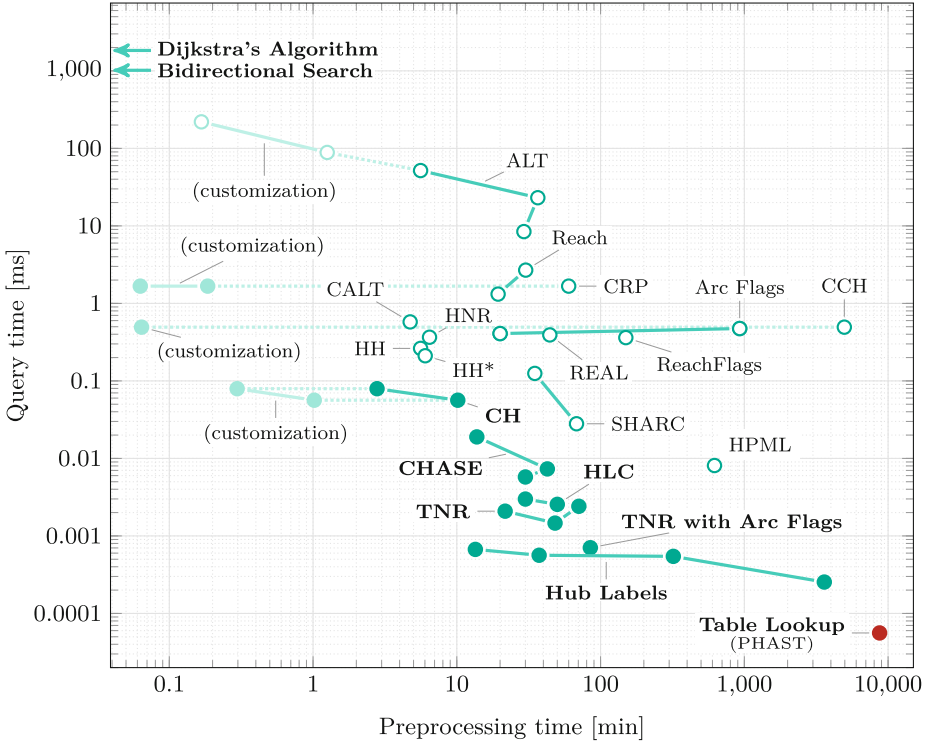


Fig. 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

model that includes turn costs. For reference, the plot includes unidirectional and bidirectional implementations of Dijkstra’s algorithm using a 4-heap. (Note that one can obtain a 20 % improvement when using a multilevel bucket queue [147].) Finally, the table-lookup figure is based on the time of a single memory access in our reference machine and the precomputation time of $|V|$ shortest path trees using PHAST [75]. Note that a machine with more than one petabyte of RAM (as required by this algorithm) would likely have slower memory access times.

Times in the plot are on a single core of an Intel X5680 3.33 GHz CPU, a mainstream server at the time of writing. Several of the algorithms in the plot were originally run on this machine [5, 75, 78, 82]; for the remaining, we divide by the following scaling factors: 2.322 for [40, 83], 2.698 for [142], 1.568 for [15], 0.837 for [107], and 0.797 for [112]. These were obtained from a benchmark (developed for this survey) that measures the time of computing several shortest path trees on the publicly available USA road network with travel times [101]. For the machines we did not have access to, we asked the authors to run the benchmark for us [112]. The benchmark is available from

<http://algo.iti.kit.edu/~pajor/survey/>, and we encourage future works to use it as a base to compare (sequential) running times with existing approaches.

The figure shows that there is no best technique. To stress this point, techniques with at least one implementation belonging to the Pareto set (considering preprocessing time, query time, and space usage) are drawn as solid circles; hollow entries are dominated. The Pareto set is quite large, with various methods allowing for a wide range of space-time trade-offs. Moreover, as we shall see when examining more realistic models, these three are not the only important criteria for real-world applications.

Table 1. Performance of various speedup techniques on Western Europe. Column *source* indicates the implementation tested for this survey.

Algorithm	Impl. source	Data structures		Queries	
		Space [GiB]	Time [h:m]	Scanned vertices	Time [μ s]
Dijkstra	[75]	0.4	–	9 326 696	2 195 080
Bidir. Dijkstra	[75]	0.4	–	4 914 804	1 205 660
CRP	[78]	0.9	1:00	2 766	1 650
Arc Flags	[75]	0.6	0:20	2 646	408
CH	[78]	0.4	0:05	280	110
CHASE	[75]	0.6	0:30	28	5.76
HLC	[82]	1.8	0:50	–	2.55
TNR	[15]	2.5	0:22	–	2.09
TNR+AF	[40]	5.4	1:24	–	0.70
HL	[82]	18.8	0:37	–	0.56
HL- ∞	[5]	17.7	60:00	–	0.25
table lookup	[75]	1 208 358.7	145:30	–	0.06

Table 1 has additional details about the methods in the Pareto set, including two versions of Dijkstra’s algorithm, one Dijkstra-based hierarchical technique (CH), three non-graph-based algorithms (TNR, HL, HLC), and two combinations (CHASE and TNR+AF). For reference, the table also includes a goal-directed technique (Arc Flags) and a separator-based algorithm (CRP), even though they are dominated by other methods. All algorithms were rerun for this survey on the reference machine (Intel X5680 3.33 GHz CPU), except those based on TNR, for which we report scaled results. All runs are single-threaded for this experiment, but note that all preprocessing algorithms could be accelerated using multiple cores (and, in some cases, even GPUs) [75, 144].

For each method, Table 1 reports the total amount of space required by all data structures (including the graph, if needed, but excluding extra information needed for path unpacking), the total preprocessing time, the number of vertices

scanned by an average query (where applicable) and the average query time. Once again, queries consist of pairs of vertices picked uniformly at random. We note that all methods tested can be parametrized (typically within a relatively narrow band) to achieve different trade-offs between query time, preprocessing time, and space. For simplicity, we pick a single “reasonable” set of parameters for each method. The only exception is HL- ∞ , which achieves the fastest reported query times but whose preprocessing is unreasonably slow.

Observe that algorithms based on any one of the approaches considered in Sect. 2 can answer queries in milliseconds or less. Separator-based (CRP), hierarchical (CH), and goal-directed (Arc Flags) methods do not use much more space than Dijkstra’s algorithm, but are three to four orders of magnitude faster. By combining hierarchy-based pruning and goal direction, CHASE improves query times by yet another order of magnitude, visiting little more than the shortest path itself. Finally, when a higher space overhead is acceptable, non-graph-based methods can be more than a million times faster than the baseline. In particular, HL- ∞ is only 5 times slower than the trivial table-lookup method, where a query consists of a single access to main memory. Note that the table-lookup method itself is impractical, since it would require more than one petabyte of RAM.

The experiments reported so far consider only random queries, which tend to be long-range. In a real system, however, most queries tend to be local. For that reason, Sanders and Schultes [223] introduced a methodology based on *Dijkstra ranks*. When running Dijkstra’s algorithm from a vertex s , the rank of a vertex u is the order in which it is taken from the priority queue. By evaluating pairs of vertices for Dijkstra ranks $2^1, 2^2, \dots, 2^{\lceil \log |V| \rceil}$ for some randomly chosen sources, all types (local, mid-range, global) of queries are evaluated. Figure 8 reports the median running times for all techniques from Table 1 (except TNR+AF, for which such numbers have never been published) for 1000 random sources and Dijkstra ranks $\geq 2^6$. As expected, algorithms based on graph searches (including Dijkstra, CH, CRP, and Arc Flags) are faster for local queries. This is not true for bounded-hop algorithms. For TNR, in particular, local queries must actually use a (significantly slower) graph-based approach. HL is more uniform overall because it never uses a graph.

Realistic Setting. Although useful, the results shown in Table 1 do not capture all features that are important for real-world systems. First, systems providing actual driving directions must account for turn costs and restrictions, which the simplified graph model ignores. Second, systems must often support multiple metrics (cost functions), such as shortest distances, avoid U-turns, avoid/prefer freeways, or avoid ferries; metric-specific data structures should therefore be as small as possible. Third, query times should be robust to the choice of cost functions: the system should not time out if an unfriendly cost function is chosen. Finally, one should be able to incorporate a new cost function quickly to account for current traffic conditions (or even user preferences).

CH has the fastest preprocessing among the algorithms in Table 1 and its queries are fast enough for interactive applications. Its performance degrades

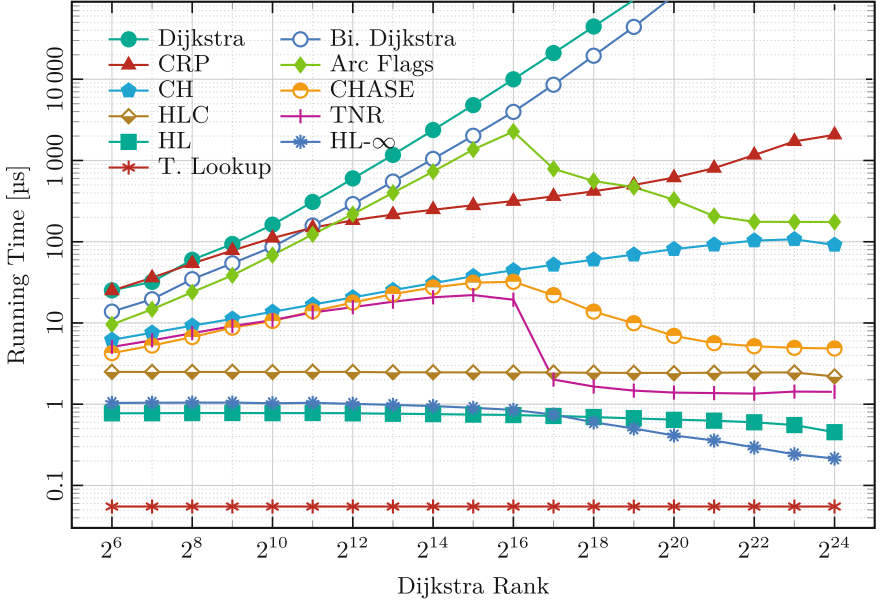


Fig. 8. Performance of speedup techniques for various Dijkstra ranks.

under realistic constraints [78], however. In contrast, CRP was developed with these constraints in mind. As explained in Sect. 2.3, it splits its preprocessing phase in two: although the initial metric-independent phase is relatively slow (as shown in Table 1), only the subsequent (and fast) metric-dependent customization phase must be rerun to incorporate a new metric. Moreover, since CRP is based on edge separators, its performance is (almost) independent of the cost function.

Table 2 (reproduced from [78]) compares CH and CRP with and without turn costs, as well as for travel distances. The instance tested is the same in Table 1, augmented by turn costs (set to 100 seconds for U-turns and zero otherwise). This simple change makes it almost as hard as fully realistic (proprietary) map data used in production systems [78]. The table reports metric-independent preprocessing and metric-dependent customization separately; “DS” refers to the data structures shared by all metrics, while “CUSTOM” refers to the additional space and time required by each individual metric. Unlike in Table 1, space consumption also includes data structures used for path unpacking. For queries, we report the time to get just the length of the shortest path (*dist*), as well as the total time to retrieve both the length and the full path (*path*). Moreover, preprocessing (and customization) times refer to multi-threaded executions on 12 cores; queries are still sequential.

As the table shows, CRP query times are very robust to the cost function and the presence of turns. Also, a new cost function can be applied in roughly 370 ms, fast enough to even support user-specific cost functions. Customization times can

Table 2. Performance of Contraction Hierarchies and CRP on a more realistic instance, using different graph representations. Preprocessing and customization times are given for multi-threaded execution on a 12-core server, while queries are run single-threaded.

Metric	Turn info	CH					CRP						
		DS		Queries			DS		Custom		Queries		
		Time [h:m]	Space [GiB]	Nmb. scans	Dist [ms]	Path [ms]	Time [h:m]	Space [GiB]	Time [s]	Space [GiB]	Nmb. scans	Dist. [ms]	Path [ms]
Dist	None	0:12	0.68	858	0.87	1.07	0:12	3.11	0.37	0.07	2942	1.91	2.49
Time	None	0:02	0.60	280	0.11	0.21	0:12	3.11	0.37	0.07	2766	1.65	1.81
	Arc-based	0:23	3.14	404	0.20	0.30	–	–	–	–	–	–	–
	Compact	0:29	1.09	1998	2.27	2.37	0:12	3.11	0.37	0.07	3049	1.67	1.85

be even reduced to 36 ms with GPUs [87], also reducing the amount of data stored in main memory by a factor of 6. This is fast enough for setting the cost function at *query time*, enabling realistic personalized driving directions on continental scale. If GPUs are not available or space consumption is an issue, one can drop the contraction-based customization. This yields customization times of about one second on a 12-core CPU, which is still fast enough for many scenarios. In contrast, CH performance is significantly worse on metrics other than travel times without turn costs.

We stress that not all applications have the same requirements. If only good estimates on travel times (and not actual paths) are needed, ignoring turn costs and restrictions is acceptable. In particular, ranking POIs according to travel times (but ignoring turn costs) already gives much better results than ranking based on geographic distances. Moreover, we note that CH has fast queries even with fully realistic turn costs. If space (for the expanded graph) is not an issue, it can still provide a viable solution to the static problem; the same holds for related methods such as HL and HLC [82]. For more dynamic scenarios, CH preprocessing can be made parallel [144] or even distributed [168]; even if run sequentially, it is fast enough for large metropolitan areas.

3.2 Applications

As discussed in Sect. 2.7, many speedup techniques can handle more than plain point-to-point shortest path computations. In particular, hierarchical techniques such as CH or CRP tend to be quite versatile, with many established extensions.

Some applications may involve more than one path between a source and a target. For example, one may want to show the user several “reasonable” paths (in addition to the shortest one) [60]. In general, these alternative paths should be short, smooth, and significantly different from the shortest path (and other alternatives). Such paths can either be computed directly as the concatenation of partial shortest paths [6, 60, 78, 173, 184] or compactly represented as a small graph [17, 174, 213]. A related problem is to compute a *corridor* [86] of paths between source and target, which allows deviations from the best route (while driving) to be handled without recomputing the entire path.

These robust routes can be useful in mobile scenarios with limited connectivity. Another useful tool to reduce communication overhead in such cases is route compression [31].

Extensions that deal with nontrivial cost functions have also been considered. In particular, one can extend CH to handle flexible arc restrictions [140] (such as height or weight limitations) or even multiple criteria [133, 138] (such as optimizing costs and travel time). Minimizing the energy consumption of electric vehicles [43, 44, 122, 152, 240, 241] is another nontrivial application, since batteries are recharged when the car is going downhill. Similarly, optimal cycling routes must take additional constraints (such as the amount of uphill cycling) into account [239].

The ability of computing many (batched) shortest paths fast enables interesting new applications. By quickly analyzing multiple candidate shortest paths, one can efficiently match GPS traces to road segments [119, 121]. Traffic simulations also benefit from acceleration techniques [183], since they must consider the likely routes taken by *all* drivers in a network. Another application is route prediction [177]: one can estimate where a vehicle is (likely) headed by measuring how good its current location is as a via point towards each candidate destination. Fast routing engines allow more locations to be evaluated more frequently, leading to better predictions [2, 121, 162, 176]. Planning placement of charging stations can also benefit from fast routing algorithms [132]. Another important application is *ride sharing* [2, 110, 139], in which one must match a ride request with the available offer in a large system, typically by minimizing drivers' detours.

Finally, batched shortest-path computations enable a wide range of point-of-interest queries [2, 99, 114, 119, 137, 179, 221, 260]. Typical examples include finding the closest restaurant to a given location, picking the best post office to stop on the way home, or finding the best meeting point for a group of friends. Typically using the bucket-based approach (cf. Sect. 2.7), fast routing engines allow POIs to be ranked according to network-based cost functions (such as travel time) rather than geographic distances. This is crucial for accuracy in areas with natural (or man-made) obstacles, such as mountains, rivers, or rail tracks. Note that more elaborate POI queries must consider concatenations of shortest paths. One can handle these efficiently using an extension of the bucket-based approach that indexes pairs of vertices instead of individual ones [2, 99].

3.3 Alternative Settings

So far, we have assumed that shortest path computations take place on a standard server with enough main memory to hold the input graph and the auxiliary data. In practice, however, it is often necessary to run (parts of) the routing algorithm in other settings, such as mobile devices, clusters, or databases. Many of the methods we discuss can be adapted to such scenarios.

Of particular interest are mobile devices, which typically are slower and (most importantly) have much less available RAM. This has motivated external memory implementation of various speedup techniques, such as ALT [150], CH [226],

and time-dependent CH [167]. CH in particular is quite practical, supporting interactive queries by compressing the routing data structures and optimizing their access patterns.

Relational databases are another important setting in practice, since they allow users to formulate complex queries on the data in SQL, a popular and expressive declarative query language [230].

Unfortunately, the table-based computational model makes it hard (and inefficient) to implement basic data structures such as graphs or even priority queues. Although some distance oracles based on geometric information could be implemented on a database [229], they are approximate and very expensive in terms of time and space, limiting their applicability to small instances. A better solution is to use HL, whose queries can very easily be expressed in SQL, allowing interactive applications based on shortest path computations entirely within a relational database [2].

For some advanced scenarios, such as time-dependent networks, the preprocessing effort increases quite a lot compared to the time-independent scenario. One possible solution is to run the preprocessing in a distributed fashion. One can achieve an almost linear speedup as the number of machine increases, for both CH [168] and CRP [116].

4 Journey Planning in Public Transit Networks

This section considers journey planning in (schedule-based) public transit networks. In this scenario, the input is given by a timetable. Roughly speaking, a timetable consists of a set of stops (such as bus stops or train platforms), a set of routes (such as bus or train lines), and a set of trips. Trips correspond to individual vehicles that visit the stops along a certain route at a specific time of the day. Trips can be further subdivided into sequences of elementary connections, each given as a pair of (origin/destination) stops and (departure/arrival) times between which the vehicle travels without stopping. In addition, footpaths model walking connections (transfers) between nearby stops.

A key difference to road networks is that public transit networks are inherently *time-dependent*, since certain segments of the network can only be traversed at specific, discrete points in time. As such, the first challenge concerns modeling the timetable appropriately in order to enable the computation of journeys, i.e., sequences of trips one can take within a transportation network. While in road networks computing a single shortest path (typically the quickest journey) is often sufficient, in public transit networks it is important to solve more involved problems, often taking several optimization criteria into account. Section 4.1 will address such modeling issues.

Accelerating queries for efficient journey planning is a long-standing problem [45, 235, 247, 248]. A large number of algorithms have been developed not only to answer basic queries fast, but also to deal with extended scenarios that incorporate delays, compute robust journeys, or optimize additional criteria, such as monetary cost.

4.1 Modeling

The first challenge is to model the timetable in order to enable algorithms that compute optimal journeys. Since the shortest-path problem is well understood in the literature, it seems natural to build a graph $G = (V, A)$ from the timetable such that shortest paths in G correspond to optimal journeys. This section reviews the two main approaches to do so (*time-expanded* and *time-dependent*), as well as the common types of problems one is interested to solve. For a more detailed overview of these topics, we refer the reader to an overview article by Müller-Hannemann et al. [203].

Time-Expanded Model. Based on the fact that a timetable consists of time-dependent *events* (e.g., a vehicle departing at a stop) that happen at *discrete* points in time, the idea of the *time-expanded* model is to build a space-time graph (often also called an event graph) [211] that “unrolls” time. Roughly speaking, the model creates a vertex for every event of the timetable and uses arcs to connect subsequent events in the direction of time flow. A basic version of the model [196, 235] contains a vertex for every departure and arrival event, with consecutive departure and arrival events connected by *connection* (or *travel*) arcs. To enable transfers between vehicles, all vertices at the same stop are (linearly, in chronological order) interlinked by *transfer* (or *waiting*) arcs. Müller-Hannemann and Weihe [204] extend the model to distinguish trains (to optimize the number of transfers taken during queries) by subdividing each connection arc by a new vertex, and then interlinking the vertices of each trip (in order of travel). Pyrga et al. [218, 219] and Müller-Hannemann and Schnee [200] extend the time-expanded model to incorporate *minimum change times* (given by the input) that are required as buffer when changing trips at a station. Their *realistic* model introduces an additional *transfer vertex* per departure event, and

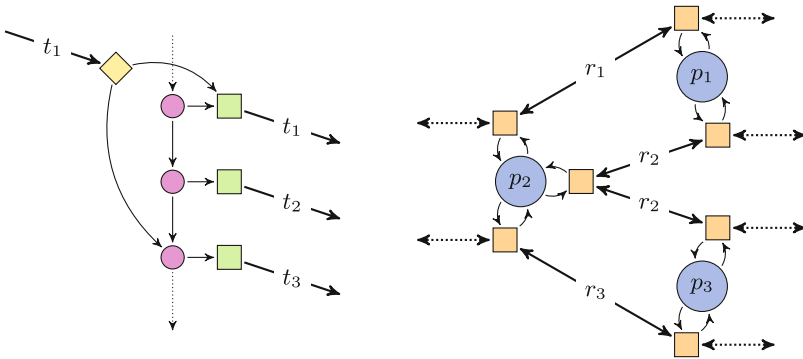


Fig. 9. Realistic time-expanded (left) and time-dependent (right) models. Different vertex types are highlighted by shape: diamond (arrival), circle (transfer) and square (departure) for the left figure; and circle (stop) and square (route) for the right figure. Connection arcs in the time-expanded model are annotated with its trips t_i , and route arcs in the time-dependent model with its routes r_i .

connects each arrival vertex to the first transfer vertex that obeys the minimum change time constraints. See Fig. 9 for an illustration. If there is a footpath from stop p_i to stop p_j , then for each arrival event at stop p_i one adds an arc to the earliest reachable transfer vertex at p_j . This model has been further engineered [90] to reduce the number of arcs that are explored “redundantly” during queries.

A timetable is usually valid for a certain period of time (up to one year). Since the timetables of different days of the year are quite similar, a space-saving technique (*compressed model*) is to consider events modulo their traffic days [202, 219].

Time-Dependent Model. The main disadvantage of the time-expanded model is that the resulting graphs are quite large [218]. For smaller graphs, the time-dependent approach (see Sect. 2.7) has been considered by Brodal and Jacob [56]. In their model, vertices correspond to stops, and an arc is added from u to v if there is at least one elementary connection serving the corresponding stops in this order. Precise departure and arrival times are encoded by the travel time function associated with the arc (u, v) . Fig. 10 shows the typical shape of a travel time function: each filled circle represents an elementary connection; the line segments (with slope -1) reflect not only the travel time, but also the waiting time until the next departure. Pyrga et al. [219] further extended this basic model to enable minimum change times by creating, for each stop p and each route that serves p , a dedicated *route vertex*. Route vertices at p are connected to a common *stop vertex* by arcs with constant cost depicting the minimum change time of p . Trips are distributed among *route arcs* that connect the subsequent route vertices of a route, as shown in Fig. 9. They also consider a model that allows arbitrary minimum change times between pairs of routes within each stop [219]. Footpaths connecting nearby stops are naturally integrated into the time-dependent model [109]. For some applications, one may merge route vertices of the same stop as long as they never connect trips such that a transfer between them violates the minimum change time [85].

Frequency-Based Model. In real-world timetables trips often operate according to specific frequencies at times of the day. For instance, a bus may run every

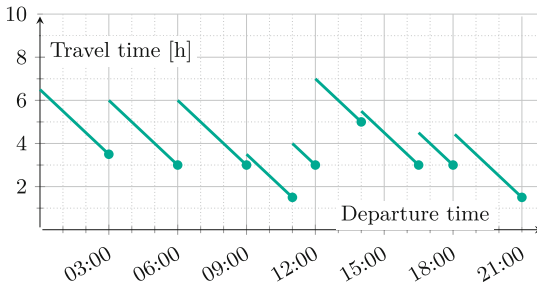


Fig. 10. Travel time function on an arc.

5 min during rush hour, and every 10 min otherwise. Bast and Storandt [27] exploit this fact in the *frequency-based model*: as in the time-dependent approach, vertices correspond to stops, and an arc between a pair of stops (u, v) is added if there is at least one elementary connection from u to v . However, instead of storing the departures of an arc explicitly, those with coinciding travel times are compressed into a set of tuples consisting of an initial departure time τ_{dep} , a time interval Δ , and a frequency f . The corresponding original departures can thus be reconstructed by computing each $\tau_{\text{dep}} + fi$ for those $i \in \mathbb{Z}_{\geq 0}$ that satisfy $\tau_{\text{dep}} + fi \leq \tau_{\text{dep}} + \Delta$. Bast and Storandt compute these tuples by covering the set of departure times by a small set of overlapping arithmetic progressions, then discarding duplicate entries (occurring after decompression) at query time [27].

Problem Variants. Most research on road networks has focused on computing the shortest path according to a given cost function (typically travel times). For public transit networks, in contrast, there is a variety of natural problem formulations.

The simplest variant is the *earliest arrival problem*. Given a source stop p_s , a target stop p_t , and a departure time τ , it asks for a journey that departs p_s no earlier than τ and arrives at p_t as early as possible. A related variant is the *range* (or *profile*) *problem* [206], which replaces the departure time by a time range (e. g. 8–10 am, or the whole day). This problem asks for a set of journeys of minimum travel time that depart within that range.

Both the earliest arrival and the range problems only consider (arrival or travel) time as optimization criterion. In public-transit networks, however, other criteria (such as the number of transfers) are just as important, which leads to the *multicriteria problem* [204]. Given source and target stops p_s, p_t and a departure time τ as input, it asks for a (maximal) Pareto set \mathcal{J} of nondominating journeys with respect to the optimization criteria considered. A journey J_1 is said to dominate journey J_2 if J_1 is better than or equal to J_2 in all criteria. Further variants of the problem relax or strengthen these domination rules [200].

4.2 Algorithms Without Preprocessing

This section discusses algorithms that can answer queries without a preprocessing phase, which makes them a good fit for dynamic scenarios that include delays, route changes, or train cancellations. We group the algorithms by the problems they are meant to solve.

Earliest Arrival Problem. Earliest arrival queries on the time-expanded model can be answered in a straightforward way by Dijkstra’s algorithm [235], in short TED (time-expanded Dijkstra). It is initialized with the vertex that corresponds to the earliest event of the source stop p_s that occurs after τ (in the realistic model, a transfer vertex must be selected). The first scanned vertex associated with the target stop p_t then represents the earliest arrival s – t journey. In the compressed time-expanded model, slight modifications to Dijkstra’s algorithm are necessary because an event vertex may appear several times on

the optimal shortest path (namely for different consecutive days). One possible solution is to use a bag of labels for each vertex as in the multicriteria variants described below. Another solution is described in Pyrga et al. [219].

On time-dependent graphs, Dijkstra’s algorithm can be augmented to compute shortest paths [65, 111], as long as the cost functions are nonnegative and FIFO [208, 209]. The only modification is that, when the algorithm scans an arc (u, v) , the arc cost is evaluated at time $\tau + \text{dist}(s, u)$. Note that the algorithm retains the label-setting property, i. e., each vertex is scanned at most once. In the time-dependent public transit model, the query is run from the stop vertex corresponding to p_s and the algorithm may stop as soon as it extracts p_t from the priority queue. The algorithm is called TDD (time-dependent Dijkstra).

Another approach is to exploit the fact that the time-expanded graph is directed and acyclic. (Note that overnight connections can be handled by unrolling the timetable for several consecutive periods.) By scanning vertices in topological order, arbitrary queries can be answered in linear time. This simple and well-known observation has been applied for journey planning by Mellouli and Suhl [191], for example. While this idea saves the relatively expensive priority queue operations of Dijkstra’s algorithm, one can do even better by not maintaining the graph structure explicitly, thus improving locality and cache efficiency. The recently developed *Connection Scan Algorithm* (CSA) [105] organizes the elementary connections of the timetable in a single array, sorted by departure time. The query then only scans this array once, which is very efficient in practice. Note that CSA requires footpaths in the input to be closed under transitivity to ensure correctness.

Range Problem. The range problem can be solved on the time-dependent model by variants of Dijkstra’s algorithm. The first variant [68, 206] maintains, at each vertex u , a travel-time function (instead of a scalar label) representing the optimal travel times from s to u for the considered time range. Whenever the algorithm relaxes an arc (u, v) , it first *links* the full travel-time function associated with u to the (time-dependent) cost function of the arc (u, v) , resulting in a function that represents the times to travel from s to v via u . This function is then *merged* into the (tentative) travel time function associated with v , which corresponds to taking the element-wise minimum of the two functions. The algorithm loses the label-setting property, since travel time functions cannot be totally ordered. As a result the algorithm may reinsert vertices into the priority queue whenever it finds a journey that improves the travel time function of an already scanned vertex.

Another algorithm [34] exploits the fact that trips depart at discrete points in time, which helps to avoid redundant work when propagating travel time functions. When it relaxes an arc, it does not consider the full function, but each of its encoded connections individually. It then only propagates the parts of the function that have improved.

The *Self-Pruning Connection Setting* algorithm (SPCS) [85] is based on the observation that *any* optimal journey from s to t has to start with one of the trips departing from s . It therefore runs, for each such trip, Dijkstra’s algorithm from s

at its respective departure time. SPCS performs these runs simultaneously using a shared priority queue whose entries are ordered by arrival time. Whenever the algorithm scans a vertex u , it checks if u has been already scanned for an associated (departing) trip with a *later* departure time (at s), in which case it prunes u . Moreover, SPCS can be parallelized by assigning different subsets of departing trips from s to different CPU cores.

Bast and Storandt [27] propose an extension of Dijkstra’s algorithm that operates on the (compressed) frequency-based model directly. It maintains with every vertex u a set of tuples consisting of a time interval, a frequency, and the travel time. Hence, a single tuple may represent multiple optimal journeys, each departing within the tuple’s time interval. Whenever the algorithm relaxes an arc (u, v) , it first extends the tuples from the bag at u with the ones stored at the arc (u, v) in the compressed graph. The resulting tentative bag of tuples (representing all optimal journeys to v via u) is then *merged* into the bag of tuples associated with v . The main challenge of this algorithm is efficiently merging tuples with incompatible frequencies and time intervals [27].

Finally, the Connection Scan Algorithm has been extended to the range problem [105]. It uses the same array of connections, ordered by departure time, as for earliest arrival queries. It still suffices to scan this array once, even to obtain optimal journeys to all stops of the network.

Multicriteria Problem. Although Pareto sets can contain exponentially many solutions (see Sect. 2.7), they are often much smaller for public transit route planning, since common optimization criteria are positively correlated. For example, for the case of optimizing earliest arrival time and number of transfers, the *Layered Dijkstra* (LD) algorithm [56, 219] is efficient. Given an upper bound K on the number of transfers, it (implicitly) copies the timetable graph into K layers, rewiring transfer arcs to point to the next higher level. It then suffices to run a time-dependent (single criterion) Dijkstra query from the lowest level to obtain Pareto sets.

In the time-expanded model, Müller-Hannemann and Schnee [200] consider the Multicriteria Label-Setting (MLS) algorithm (cf. Sect. 2.7) to optimize arrival time, ticket cost, and number of transfers. In the time-dependent model, Pyrga et al. [219] compute Pareto sets of journeys for arrival time and number of transfers. Disser et al. [109] propose three optimizations to MLS that reduce the number of queue operations: hopping reduction, label forwarding, and dominance by early results (or *target pruning*). Bast and Storandt [27] extend the frequency-based range query algorithm to also include number of transfers as criterion.

A different approach is *RAPTOR* (Round-bAsed Public Transit Optimized Router) [92]. It is explicitly developed for public transit networks and its basic version optimizes arrival time and the number of transfers taken. Instead of using a graph, it organizes the input as a few simple arrays of trips and routes. Essentially, RAPTOR is a dynamic program: it works in rounds, with round i computing earliest arrival times for journeys that consist of exactly i transfers. Each round takes as input the stops whose arrival time improved in the previous round (for the first round this is only the source stop). It then *scans* the routes

served by these stops. To scan route r , RAPTOR traverses its stops in order of travel, keeping track of the earliest possible trip (of r) that can be taken. This trip may improve the tentative arrival times at subsequent stops of route r . Note that RAPTOR scans each route at most once per round, which is very efficient in practice (even faster than Dijkstra’s algorithm with a single criterion). Moreover, RAPTOR can be parallelized by distributing non-conflicting routes to different CPU cores. It can also be extended to handle range queries (rRAPTOR) and additional optimization criteria (McRAPTOR). Note that, like CSA, RAPTOR also requires footpaths in the input to be closed under transitivity.

Trip-Based Routing [256] accelerates RAPTOR by executing a BFS-like search on a network of trips and precomputed *sensible* transfers.

4.3 Speedup Techniques

This section presents an overview of preprocessing-based speedup techniques for journey planning in public transit networks. A natural (and popular) approach is to adapt methods that are effective on road networks (see Fig. 7). Unfortunately, the speedups observed in public transit networks are several orders of magnitude lower than in road networks. This is to some extent explained by the quite different structural properties of public transit and road networks [22]. For example, the neighborhood of a stop can be much larger than the number of road segments incident to an intersection. Even more important is the effect of the inherent time-dependency of public transit networks. Thus, developing efficient preprocessing-based methods for public transit remains a challenge.

Some road network methods were tested on public transit graphs without performing realistic queries (i. e., according to one of the problems from Sect. 4.1). Instead, such studies simply perform point-to-point queries on public-transit graphs. In particular, Holzer et al. [161] evaluate basic combinations of bidirectional search, goal directed search, and Geometric Containers on a simple stop graph (with average travel times). Bauer et al. [41] also evaluated bidirectional search, ALT, Arc Flags, Reach, REAL, Highway Hierarchies, and SHARC on time-expanded graphs. Core-ALT, CHASE, and Contraction Hierarchies have also been evaluated on time-expanded graphs [40].

A Search.* On public transit networks, basic A* search has been applied to the time-dependent model [109, 219]. In the context of multicriteria optimization, Disser et al. [109] determine lower bounds for each vertex u to the target stop p_t (before the query) by running a backward search (from p_t) using the (constant) lower bounds of the travel time functions as arc cost.

ALT. The (unidirectional) ALT [148] algorithm has been adapted to both the time-expanded [90] and the time-dependent [207] models for computing earliest arrival queries. In both cases, landmark selection and distance precomputation is performed on an auxiliary stop graph, in which vertices correspond to stops and an arc is added between two stops p_i, p_j if there is an elementary connection from p_i to p_j in the input. Arc costs are lower bounds on the travel time between their endpoints.

Geometric Containers. Geometric containers [235, 251] have been extensively tested on the time-expanded model for computing earliest arrival queries. In fact, they were developed in the context of this model. As mentioned in Sect. 2, bounding boxes perform best [251].

Arc Flags and SHARC. Delling et al. [90] have adapted Arc Flags [157, 178] to the time-expanded model as follows. First, they compute a partition on the stop graph (defined as in ALT). Then, for each boundary stop p of cell C , and each of its arrival vertices, a backward search is performed on the time-expanded graph. The authors observe that public transit networks have many paths of equal length between the same pair of vertices [90], making the choice of tie-breaking rules important. Furthermore, Delling et al. [90] combine Arc Flags, ALT, and a technique called *Node Blocking*, which avoids exploring multiple arcs from the same route.

SHARC, which combines Arc Flags with shortcuts [39], has been tested on the time-dependent model with earliest arrival queries by Delling [72]. Moreover, Arc Flags with shortcuts for the Multi-Label-Setting algorithm (MLS) have been considered for computing full (i. e., using strict domination) Pareto sets using arrival time and number of transfers as criteria [47]. In time-dependent graphs, a flag must be set if its arc appears on a shortest path toward the corresponding cell at least once during the time horizon [72]. For better performance, one can use different sets of flags for different time periods (e. g., every two hours). The resulting total speedup is still below 15, from which it is concluded that “accelerating time-dependent multicriteria timetable information is harder than expected” [47]. Slight additional speedups can be obtained if one restricts the search space to only those solutions in the Pareto set for which the travel time is within an interval defined by the earliest arrival time and some upper bound. Berger et al. [49] observed that in such a scenario optimal substructure in combination with lower travel time bounds can be exploited and yield additional pruning during search. It is worth noting that this method does not require any preprocessing and is therefore well-suited for a dynamic scenario.

Overlay Graphs. To accelerate earliest arrival queries, Schulz et al. [235] compute single-level overlays between “important” hub stations in the time-expanded model, with importance values given as input. More precisely, given a subset of important stations, the overlay graph consists of *all* vertices (events) that are associated with these stations. Edges in the overlay are computed such that distances between any pair of vertices (events) are preserved. Extending this approach to overlay graphs over multiple levels of hub stations (selected by importance or degree) results in speedups of about 11 [236].

Separator-based techniques. Strasser and Wagner [242] combine the Connection Scan Algorithm [105] with ideas of customizable route planning (CRP) [78] resulting in the Accelerated Connection Scan Algorithm (ACSA). It is designed for both earliest arrival and range queries. ACSA first computes a multilevel partition of

stops, minimizing the number of elementary connections with endpoints in different cells. Then, it precomputes for each cell the partial journeys (transit connections) that cross the respective cell. For queries, the algorithm essentially runs CSA restricted to the elementary connections of the cells containing the source or target stops, as well as transit connections of other (higher-level) cells. As shown in Sect. 4.5, it achieves excellent query and preprocessing times on country-sized instances.

Contraction Hierarchies. The Contraction Hierarchies algorithm [142] has been adapted to the realistic time-dependent model with minimum change times for computing earliest arrival and range queries [136]. It turns out that simply applying the algorithm to the route model graph results in too many shortcuts to be practical. Therefore, contraction is performed on a condensed graph that contains only a single vertex per stop. Minimum change times are then ensured by the query algorithm, which must maintain multiple labels per vertex.

Transfer Patterns. A speedup technique specifically developed for public transit networks is called *Transfer Patterns* [24]. It is based on the observation that many optimal journeys share the same transfer pattern, defined as the sequence of stops where a transfer occurs. Conceptually, these transfer patterns are pre-computed using range queries for all pairs of stops and departure times. At query time, a query graph is built as the union of the transfer patterns between the source and target stops. The arcs in the query graph represent direct connections between stops (without transfers), and can be evaluated very fast. Dijkstra’s algorithm (or MLS) is then applied to this much smaller query graph.

If precomputing transfer patterns between *all* pairs of stops is too expensive, one may resort to the following two-level approach. It first selects a subset of (important) hub stops. From the hubs, global transfer patterns are precomputed to all other stops. For the non-hubs, local transfer patterns are computed only towards relevant hub stops. This approach is similar to TNR, but the idea is applied asymmetrically: transfer patterns are computed from all stops to the hub stops, and from the hub stops to everywhere. If preprocessing is still impractical, one can restrict the local transfer patterns to at most three legs (two transfers). Although this restriction is heuristic, the algorithm still almost always finds the optimal solution in practice, since journeys requiring more than two transfers to reach a hub station are rare [24].

TRANSIT. Finally, Transit Node Routing [28, 30, 224] has been adapted to public transit journey planning in [14]. Preprocessing of the resulting *TRANSIT* algorithm uses the (small) stop graph to determine a set of transit nodes (with a similar method as in [28]), between which it maintains a distance table that contains sets of journeys with minimal travel time (over the day). Each stop p maintains, in addition, a set of access nodes $A(p)$, which is computed on the time-expanded graph by running local searches from each departure event of p toward the transit stops. The query then uses the access nodes of p_s and p_t and

the distance table to resolve global requests. For local requests, it runs goal-directed A* search. Queries are slower than for Transfer Patterns.

4.4 Extended Scenarios

Besides computing journeys according to one of the problems from Sect. 4.1, extended scenarios (such as incorporating delays) have been studied as well.

Uncertainty and Delays. Trains, buses and other means of transport are often prone to delays in the real world. Thus, handling delays (and other sources of uncertainty) is an important aspect of a practical journey planning system. Firmani et al. [125] recently presented a case study for the public transport network of the metropolitan area of Rome. They provide strong evidence that computing journeys according to the published timetable often fails to deliver optimal or even high-quality solutions. However, incorporating real-time GPS location data of vehicles into the journey planning algorithm helps improve the journey quality (e.g., in terms of the experienced delay) [13, 84].

Müller-Hannemann and Schnee [201] consider the online problem where delays, train cancellations, and extra trains arrive as a continuous stream of information. They present an approach which quickly updates the time-expanded model to enable queries according to current conditions. Delling et al. [74] also discuss updating the time-dependent model and compare the required effort with the time-expanded model. Cionini et al. [63] propose a new graph-based model which is tailored to handle dynamic updates, and they experimentally show its effectiveness in terms of both query and update times. Berger et al. [48] propose a realistic stochastic model that predicts how delays propagate through the network. In particular, this model is evaluated using real (delay) data from Deutsche Bahn. Bast et al. [25] study the robustness of Transfer Patterns with respect to delays. They show that the transfer patterns computed for a scenario without any delays give optimal results for 99 % of queries, even when large and area-wide (random) delays are injected into the networks.

Disser et al. [109] and Delling et al. [93] study the computation of *reliable* journeys via multicriteria optimization. The reliability of a transfer is defined as a function of the available buffer time for the transfer. Roughly speaking, the larger the buffer time, the more likely it is that the transfer will be successful. According to this notion, transfers with a high chance of success are still considered reliable even if there is no backup alternative in case they fail.

To address this issue, Dibbelt et al. [105] minimize the *expected arrival time* (with respect to a simple model for the probability that a transfer breaks). Instead of journeys, their method (which is based on the CSA algorithm) outputs a *decision graph* representing optimal instructions to the user at each point of their journey, including cases in which a connecting trip is missed. Interestingly, minimizing the expected arrival time implicitly helps minimizing the number of transfers, since each “unnecessary” transfer introduces additional uncertainty, hurting the expected arrival time.

Finally, Goerigk et al. [146] study the computation of *robust* journeys, considering both strict robustness (i. e., computing journeys that are always feasible for a given set of delay scenarios) and light robustness (i. e., computing journeys that are most reliable when given some extra slack time). While strict robustness turns out to be too conservative in practice, the notion of light robustness seems more promising. *Recoverable robust* journeys (which can always be updated when delays occur) have recently been considered in [145]. A different, new robustness concept has been proposed by Böhmová et al. [51]. In order to propose solutions that are robust for typical delays, past observations of real traffic situations are used. Roughly speaking, a route is more robust the better it has performed in the past under different scenarios.

Night Trains. Gunkel et al. [153] have considered the computation of overnight train journeys, whose optimization goals are quite different from regular “day-time” journeys. From a customer’s point of view, the primary objective is usually to have a reasonably long sleeping period. Moreover, arriving too early in the morning at the destination is often not desired. Gunkel et al. present two approaches to compute overnight journeys. The first approach explicitly enumerates all overnight trains (which are given by the input) and computes, for each such train, the optimal feeding connections. The second approach runs multicriteria search with sleeping time as a maximization criterion.

Fares. Müller-Hannemann and Schnee [199] have analyzed several pricing schemes, integrating them as an optimization criterion (cost) into MOTIS, a multicriteria search algorithm that works on the time-expanded model. In general, however, optimizing exact monetary cost is a challenging problem, since real-world pricing schemes are hard to capture by a mathematical model [199].

Delling et al. [92] consider computing Pareto sets of journeys that optimize fare zones with the McRAPTOR algorithm. Instead of using (monetary) cost as an optimization criterion directly, they compute all nondominated journeys that traverse different combinations of fare zones, which can then be evaluated by cost in a quick postprocessing step.

Guidebook Routing. Bast and Storandt [26] introduce *Guidebook Routing*, where the user specifies only source and target stops, but neither a day nor a time of departure. The desired answer is then a set of routes, each of which is given by a sequence of train or bus numbers and transfer stations. For example, an answer may read like *take bus number 11 towards the bus stop at X, then change to bus number 13 or 14 (whichever comes first) and continue to the bus stop at Y*. Guidebook routes can be computed by first running a multicriteria range query, and then extracting from the union of all Pareto-optimal time-dependent paths a subset of routes composed by arcs which are most frequently used. The Transfer Patterns algorithm lends itself particularly well to the computation of such guidebook routes. For practical guidebook routes (excluding “exotic” connections at particular times), the preprocessing space and query times of Transfer Patterns can be reduced by a factor of 4 to 5.

4.5 Experiments and Comparison

This section compares the performance of some of the journey planning algorithms discussed in this section. As in road networks, all algorithms have been carefully implemented in C++ using mostly custom-built data structures.

Table 3 summarizes the results. Running times are obtained from a sequential execution on one core of a dual 8-core Intel Xeon E5-2670 machine clocked at 2.6 GHz with 64 GiB of DDR3-1600 RAM. The exceptions are Transfer Patterns and Contraction Hierarchies, for which we reproduce the values reported in the original publication (obtained on a comparable machine).

For each algorithm, we report the instance on which it has been evaluated, as well as its total number of elementary connections (a proxy for size) and the number of consecutive days covered by the connections. Unfortunately, realistic benchmark data of country scale (or larger) has not been widely available to the research community. Some metropolitan transit agencies have recently started making their timetable data publicly available, mostly using the General Transit Feed format². Still, research groups often interpret the data differently, making it hard to compare the performance of different algorithms. The largest metropolitan instance currently available is the full transit network of London³. It contains approximately 21 thousand stops, 2.2 thousand routes, 133 thousand trips, 46 thousand footpaths, and 5.1 million elementary connections for one full day. We therefore use this instance for the evaluation of most algorithms. The instances representing Germany and long-distance trains in Europe are generated in a similar way, but from proprietary data.

The table also contains the preprocessing time (where applicable), the average number of label comparisons per stop, the average number of journeys computed by the algorithm, and its running time in milliseconds. Note that the number of journeys can be below 1 because some stops are unreachable for certain late departure times. References indicate the publications from which the figures are taken (which may differ from the first publication); TED was run by the authors for this survey. (Our TED implementation uses a single-level bucket queue [104] and stops as soon as a vertex of the target stop has been extracted.) The columns labeled “criteria” indicate whether the algorithm minimizes arrival time (arr), number of transfers (tran), fare zones (fare), reliability (rel), and whether it computes range queries (rng) over the full timetable period of 1, 2, or 7 days. Methods with multiple criteria compute Pareto sets.

Among algorithms without preprocessing, we observe that those that do not use a graph (RAPTOR and CSA) are consistently faster than their graph-based counterparts. Moreover, running Dijkstra’s algorithm on the time-expanded graph model (TED) is significantly slower than running it on the time-dependent graph model (TDD), since time-expanded graphs are much larger. For earliest arrival queries on metropolitan areas, CSA is the fastest algorithm without preprocessing, but preprocessing-based methods (such as Transfer Patterns) can

² <https://developers.google.com/transit/gtfs/>.

³ <http://data.london.gov.uk/>.

Table 3. Performance of various public transit algorithms on random queries. For each algorithm, the table indicates the implementation tested (which may not be the publication introducing the algorithm), the instance it was tested on, its total number of elementary connections (in millions) as well as the number of consecutive days they cover. A “p” indicates that the timetable is periodic (with a period of one day). The table then shows the criteria that are optimized (a subset of arrival times, transfers, full range, fares, and reliability), followed by total preprocessing time, average number of comparisons per stop, average number of journeys in the Pareto set, and average query times in milliseconds. Missing entries either do not apply (–) or are well-defined but not available (n/a).

Algorithm	Impl.	INPUT			CRITERIA							QUERY		
		Name	Conn. [10 ⁶]	Dy.	Arr.	Tran.	Rng.	Fare	Rel.	Prep [h]	Comp./ stop	jn	Time [ms]	
TED		London	5.1	1	●	o	o	o	o	–	50.6	0.9	44.8	
TDD	[93]	London	5.1	1	●	o	o	o	o	–	7.4	0.9	11.0	
CH	[136]	Europe (Ing)	1.7	p	●	o	o	o	o	< 0.1	< 0.1	n/a	0.3	
CSA	[105]	London	4.9	1	●	o	o	o	o	–	26.6	n/a	2.0	
ACSA	[242]	Germany	46.2	2	●	o	o	o	o	–	n/a	n/a	8.7	
T. Patterns	[27]	Germany	90.4	7	●	o	o	o	o	541	–	1.0	0.4	
LD	[93]	London	5.1	1	●	●	o	o	o	–	15.6	1.8	28.7	
MLS	[93]	London	5.1	1	●	●	o	o	o	–	23.7	1.8	50.0	
RAPTOR	[93]	London	5.1	1	●	●	o	o	o	–	10.9	1.8	5.4	
T. Patterns	[27]	Germany	90.4	7	●	●	o	o	o	566	–	2.0	0.8	
CH	[136]	Europe (Ing)	1.7	p	●	o	●	o	o	< 0.1	< 0.1	n/a	3.7	
SPCS	[105]	London	4.9	1	●	o	●	o	o	–	372.5	98.2	843.0	
CSA	[105]	London	4.9	1	●	o	●	o	o	–	436.9	98.2	161.0	
ACSA	[242]	Germany	46.2	2	●	o	●	o	o	8	n/a	n/a	171.0	
T. Patterns	[27]	Germany	90.4	7	●	o	●	o	o	541	–	121.2	22.0	
rRAPTOR	[105]	London	4.9	1	●	●	●	o	o	–	1634.0	203.4	922.0	
CSA	[105]	London	4.9	1	●	●	●	o	o	–	3824.9	203.4	466.0	
T. Patterns	[27]	Germany	90.4	7	●	●	●	o	o	566	–	226.0	39.6	
MLS	[93]	London	5.1	1	●	●	o	●	o	–	818.2	8.8	304.2	
McRAPTOR	[93]	London	5.1	1	●	●	o	●	o	–	277.5	8.8	100.9	
MLS	[93]	London	5.1	1	●	●	o	o	●	–	286.6	4.7	239.8	
McRAPTOR	[93]	London	5.1	1	●	●	o	o	●	–	89.6	4.7	71.9	

be even faster. For longer-range transit networks, preprocessing-based methods scale very well. CH takes 210 seconds to preprocess the long-distance train connections of Europe, while ACSA takes 8 hours to preprocess the full transit network of Germany. Transfer Patterns takes over 60 times longer to preprocess (a full week of) the full transit network of Germany, but has considerably lower query times.

For multicriteria queries, RAPTOR is about an order of magnitude faster than Dijkstra-based approaches like LD and MLS. RAPTOR is twice as fast as TDD, while computing twice as many journeys on average. Adding further criteria (such as fares and reliability) to MLS and RAPTOR increases the Pareto set, but performance is still reasonable for metropolitan-sized networks. Thanks to preprocessing, Transfer Patterns has the fastest queries overall, by more than an

order of magnitude. Note that in public transit networks the optimization criteria are often positively correlated (such as arrival time and number of transfers), which keeps the Pareto sets at a manageable size. Still, as the number of criteria increases, exact real-time queries become harder to achieve.

The reported figures for Transfer Patterns are based on preprocessing leveraging the frequency-based model with traffic days compression, which makes quadratic (in the number of stops) preprocessing effort feasible. Consequently, hub stops and the three-leg heuristic are not required, and the algorithm is guaranteed to find the optimal solution. The data produced by the preprocessing is shown to be robust against large and area-wide delays, resulting in much less than 1 % of suboptimal journeys [25] (not shown in the table).

For range queries, preprocessing-based techniques (CH, ACSA, Transfer Patterns) scale better than CSA or SPCS. For full multicriteria range queries (considering transfers), Transfer Patterns is by far the fastest method, thanks to preprocessing. Among search-based methods, CSA is faster than rRAPTOR by a factor of two, although it does twice the amount of work in terms of label comparisons. Note, however, that while CSA cannot scale to smaller time ranges by design [105], the performance of rRAPTOR depends linearly on the number of journeys departing within the time range [92]. For example, for 2-hour range queries rRAPTOR computes 15.9 journeys taking only 61.3 ms on average [93] (not reported in the table). Guidebook routes covering about 80 % of the optimal results (for the full period) can be computed in a fraction of a millisecond [26].

5 Multimodal Journey Planning

We now consider journey planning in a multimodal scenario. Here, the general problem is to compute journeys that *reasonably* combine different modes of transportation by a *holistic* algorithmic approach. That is, not only does an algorithm consider each mode of transportation in isolation, but it also optimizes the choice (and sequence) of transportation modes in some integrated way. Transportation modes that are typically considered include (unrestricted) walking, (unrestricted) car travel, (local and long-distance) public transit, flight networks, and rental bicycle schemes. We emphasize that our definition of “multimodal” requires some diversity from the transportation modes, i. e., both unrestricted and schedule-based variants should be considered by the algorithm. For example, journeys that only use buses, trams, or trains are not truly multimodal (according to our definition), since these transportation modes can be represented as a single public transit schedule and dealt with by algorithms from Sect. 4.

In fact, considering modal transfers explicitly by the algorithm is crucial in practice, since the solutions it computes must be *feasible*, excluding sequences of transportation modes that are impossible for the user to take (such as a private car between train rides). Ideally, even user preferences should be respected. For example, some users may prefer taxis over public transit at certain parts of the journey, while others may not.

A general approach to obtain a multimodal network is to first build an individual graph for each transportation mode, then merge them into a single multimodal graph with *link arcs* (or vertices) added to enable modal transfers [89,210,258]. Typical examples [89,210] model car travel and walking as time-independent (static) graphs, public transit networks using the realistic time-dependent model [219], and flight networks using a dedicated flight model [91]. Beyond that, Kirchler et al. [169,170] compute multimodal journeys in which car travel is modeled as a time-dependent network in order to incorporate historic data on rush hours and traffic congestion (see Sect. 2.7 for details).

Overview. The remainder of this section discusses three different approaches to the multimodal problem. The first (Sect. 5.1) considers a combined cost function of travel time with some penalties to account for modal transfers. The second approach (Sect. 5.2) uses the label-constrained shortest path problem to obtain journeys that explicitly include (or exclude) certain sequences of transportation modes. The final approach (Sect. 5.3) computes Pareto sets of multimodal journeys using a carefully chosen set of optimization criteria that aims to provide diverse (regarding the transportation modes) alternative journeys.

5.1 Combining Costs

To aim for journeys that reasonably combine different transport modes, one may use penalties in the objective function of the algorithm. These penalties are often considered as a linear combination with the primary optimization goal (typically travel time). Examples for this approach include Aifadopoulou et al. [10], who present a linear program that computes multimodal journeys. The TRANSIT algorithm [14] also uses a linear utility function and incorporates travel time, ticket cost, and “inconvenience” of transfers. Finally, Modesti and Sciomachen [195] consider a combined network of unrestricted walking, unrestricted car travel, and public transit, in which journeys are optimized according to a linear combination of several criteria, such as cost and travel time. Moreover, their utility function incorporates user preferences on the transportation modes.

5.2 Label-Constrained Shortest Paths

The *label-constrained shortest paths* [21] approach computes journeys that explicitly obey certain constraints on the modes of transportation. It defines an alphabet Σ of modes of transportation and labels each arc of the graph by the appropriate symbol from Σ . Then, given a language L over Σ as additional input to the query, any journey (path) must obey the constraints imposed by the language L , i.e., the concatenation of the labels along the path must satisfy L . The problem of computing *shortest* label-constrained paths is tractable for *regular* languages [21], which suffice to model reasonable transport mode constraints in multimodal journey planning [18,20]. Even restricted classes of

regular languages can be useful, such as those that impose a hierarchy of transport modes [50, 89, 169, 170, 210, 258] or Kleene languages that can only globally exclude (and include) certain transport modes [140].

Barrett et al. [21] have proven that the label-constrained shortest path problem is solvable in deterministic polynomial time. The corresponding algorithm, called *label-constrained shortest path problem Dijkstra* (LCSPD), first builds a product network \mathbf{G} of the input (the multimodal graph) and the (possibly non-deterministic) finite automaton that accepts the regular language L . For given source and target vertices s, t (referring to the original input), the algorithm determines origin and destination sets of product vertices from \mathbf{G} , containing those product vertices that refer to s/t and an initial/final state of the automaton. Dijkstra’s algorithm is then run on \mathbf{G} between these two sets of product vertices. In a follow-up experimental study, Barrett et al. [20] evaluate this algorithm using linear regular languages, a special case.

Basic speedup techniques, such as bidirectional search [67], A* [156], and heuristic A* [237] have been evaluated in the context of multimodal journey planning in [159] and [19]. Also, Pajor [210] combines the LCSPD algorithm with time-dependent Dijkstra [65] to compute multimodal journeys that contain a time-dependent subnetwork. He also adapts and analyzes bidirectional search [67], ALT [148], Arc Flags [157, 178], and shortcuts [249] with respect to LCSPD.

Access-Node Routing. The *Access-Node Routing* (ANR) [89] algorithm is a speedup technique for the label-constrained shortest path problem (LCSPD). It handles *hierarchical languages*, which allow constraints such as restricting walking and car travel to the beginning and end of the journey. It works similarly to Transit Node Routing [28–30, 224] and precomputes for each vertex u of the road (walking and car) network its relevant set of entry (and exit) points (*access nodes*) to the public transit and flight networks. More precisely, for any shortest path P originating from vertex u (of the road network) that also uses the public transit network, the first vertex v of the public transit network on P must be an access node of u . The query may skip over the road network by running a multi-source multi-target algorithm on the (much smaller) transit network between the access nodes of s and t , returning the journey with earliest combined arrival time.

The *Core-Based ANR* [89] method further reduces preprocessing space and time by combining ANR with contraction. As in Core-ALT [40, 88], it precomputes access nodes only for road vertices in a much smaller core (overlay) graph. The query algorithm first (quickly) determines the relevant core vertices of s and t (i. e., those covering the branches of the shortest path trees rooted at s and t), then runs a multi-source multi-target ANR query between them.

Access-Node Routing has been evaluated on multimodal networks of intercontinental size that include walking, car travel, public transit, and flights. Queries run in milliseconds, but preprocessing time strongly depends on the density of the public transit and flight networks [89]. Moreover, since the regular language

is used during preprocessing, it can no longer be specified at query time without loss of optimality.

State-Dependent ALT. Another multimodal speedup technique for LCSP is *State-Dependent ALT* (SDALT) [170]. It augments the ALT algorithm [148] to overcome the fact that lower bounds from a vertex u may depend strongly on the current state q of the automaton (expressing the regular language) with which u is scanned. SDALT thus uses the automaton to precompute state-dependent distances, providing lower bound values per vertex *and* state. For even better query performance, SDALT can be extended to use more aggressive (and potentially incorrect) bounds to guide the search toward the target, relying on a label-correcting algorithm (which may scan vertices multiple times) to preserve correctness [169]. SDALT has been evaluated [169, 170] on a realistic multimodal network covering the Île-de-France area (containing Paris) incorporating rental and private bicycles, public transit, walking, and a time-dependent road network for car travel. The resulting speedups are close to 30. Note that SDALT, like ANR, also predetermines the regular language constraints during preprocessing.

Contraction Hierarchies. Finally, Dibbelt et al. [106] have adapted Contraction Hierarchies [142] to LCSP, handling arbitrary mode *sequence* constraints. The resulting User-Constrained Contraction Hierarchies (UCCH) algorithm works by (independently) only contracting vertices whose incident arcs belong to the same modal subnetwork. All other vertices are kept uncontracted. The query algorithm runs in two phases. The first runs a regular CH query in the subnetworks given as initial or final transport modes of the sequence constraints until the uncontracted *core graph* is reached. Between these entry and exit vertices, the second phase then runs a regular LCSP-Dijkstra restricted to the (much smaller) core graph. Query performance of UCCH is comparable to Access-Node Routing, but with significantly less preprocessing time and space. Also, in contrast to ANR, UCCH also handles arbitrary mode sequence constraints at query time.

5.3 Multicriteria Optimization

While label constraints are useful to define feasible journeys, computing the (single) shortest label-constrained path has two important drawbacks. First, in order to define the constraints, users must know the characteristics of the particular transportation network; second, alternative journeys that combine the available transportation modes differently are not computed. To obtain a set of diverse alternatives, multicriteria optimization has been considered.

The criteria optimized by these methods usually include arrival time and, for each mode of transportation, some mode-dependent optimization criterion [23, 73]. The resulting Pareto sets will thus contain journeys with different usage of the available transportation modes, from which users can choose their favorites.

Delling et al. [73] consider networks of metropolitan scale and use the following criteria as proxies for “convenience”: number of transfers in public transit,

walking duration for the pedestrian network, and monetary cost for taxis. They observe that simply applying the MLS algorithm [155, 187, 196, 243] to a comprehensive multimodal graph turns out to be slow, even when partial contraction is applied to the road and pedestrian networks, as in UCCH [106]. To get better query performance, they extend RAPTOR [92] to the multimodal scenario, which results in the *multimodal multicriteria RAPTOR* algorithm (MCR) [73]. Like RAPTOR, MCR operates in rounds (one per transfer) and computes Pareto sets of optimal journeys with exactly i transfers in round i . It does so by running, in each round, a dedicated subalgorithm (RAPTOR for public transit; MLS for walking and taxi) which obtains journeys with the respective transport mode as their last leg.

Since with increasing number of optimization criteria the resulting Pareto sets tend to get very large, Delling et al. identify the most significant journeys in a quick postprocessing step by a scoring method based on fuzzy logic [259]. For faster queries, MCR-based heuristics (which relax domination during the algorithm) successfully find the most significant journeys while avoiding the computation of insignificant ones in the first place.

Bast et al. [23] use MLS with contraction to compute multimodal multicriteria journeys at a metropolitan scale. To identify the significant journeys of the Pareto set, they propose a method called *Types aNd Thresholds* (TNT). The method is based on a set of simple *axioms* that summarize what most users would consider as unreasonable multimodal paths. For example, if one is willing to take the car for a large fraction of the trip, one might as well take it for the whole trip. Three types of reasonable trips are deduced from the axioms: (1) only car, (2) arbitrarily much transit and walking with no car, and (3) arbitrarily much transit with little or no walking and car. With a concrete threshold for “little” (such as 10 min), the rules can then be applied to filter the reasonable journeys. As in [73], filtering can be applied during the algorithm to prune the search space and reduce query time. The resulting sets are fairly robust with respect to the choice of threshold.

6 Final Remarks

The last decade has seen astonishing progress in the performance of shortest path algorithms on transportation networks. For routing in road networks, in particular, modern algorithms can be up to seven orders of magnitude faster than standard solutions. Successful approaches exploit different properties of road networks that make them easier to deal with than general graphs, such as goal direction, a strong hierarchical structure, and the existence of small separators. Although some early acceleration techniques relied heavily on geometry (road networks are after all embedded on the surface of the Earth), no current state-of-the-art algorithm makes explicit use of vertex coordinates (see Table 1). While one still sees the occasional development (and publication) of geometry-based algorithms they are consistently dominated by established techniques. In particular, the recent Arterial Hierarchies [261] algorithm is compared

to CH (which has slightly slower queries), but not to other previously published techniques (such as CHASE, HL, and TNR) that would easily dominate it. This shows that results in this rapidly-evolving area are often slow to reach some communities; we hope this survey will help improve this state of affairs.

Note that experiments on real data are very important, as properties of production data are not always accurately captured by simplified models and folklore assumptions. For example, the common belief that an algorithm can be augmented to include turn penalties without significant loss in performance turned out to be wrong for CH [76].

Another important lesson from recent developments is that careful engineering is essential to unleash the full computational power of modern computer architectures. Algorithms such as CRP, CSA, HL, PHAST, and RAPTOR, for example, achieve much of their good performance by carefully exploiting locality of reference and parallelism (at the level of instructions, cores, and even GPUs).

The ultimate validation of several of the approaches described here is that they have found their way into systems that serve millions of users every day. Several authors of papers cited in this survey have worked on routing-related projects for companies like Apple, Esri, Google, MapBox, Microsoft, Nokia, PTV, TeleNav, TomTom, and Yandex. Although companies tend to be secretive about the actual algorithms they use, in some cases this is public knowledge. TomTom uses a variant of Arc Flags with shortcuts to perform time-dependent queries [231]. Microsoft’s Bing Maps⁴ use CRP for routing in road networks. OSRM [185], a popular route planning engine using OpenStreetMap data, uses CH for queries. The Transfer Patterns [24] algorithm has been in use for public-transit journey planning on Google Maps⁵ since 2010. RAPTOR is currently in use by OpenTripPlanner⁶.

These recent successes do not mean that all problems in this area are solved. The ultimate goal, a worldwide multimodal journey planner, has not yet been reached. Systems like Rome2Rio⁷ provide a simplified first step, but a more useful system would take into account real-time traffic and transit information, historic patterns, schedule constraints, and monetary costs. Moreover, all these elements should be combined in a personalized manner. Solving such a general problem efficiently seems beyond the reach of current algorithms. Given the recent pace of progress, however, a solution may be closer than expected.

⁴ <http://www.bing.com/blogs/site-blogs/b/maps/archive/2012/01/05/bing-maps-new-routing-engine.aspx>.

⁵ <http://www.google.com/transit>.

⁶ <http://opentripplanner.com>.

⁷ <http://www.rome2rio.com>.

References

1. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: VC-dimension and shortest path algorithms. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 690–699. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-22006-7_58](https://doi.org/10.1007/978-3-642-22006-7_58)
2. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: HLDB: Location-based services in databases. In: Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS 2012), pp. 339–348. ACM Press 2012. Best Paper Award
3. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway dimension and provably efficient shortest path algorithms. Technical report MSR-TR-2013-91, Microsoft Research (2013)
4. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A hub-based labeling algorithm for shortest paths in road networks. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 230–241. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-20662-7_20](https://doi.org/10.1007/978-3-642-20662-7_20)
5. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Hierarchical hub labelings for shortest paths. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 24–35. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-33090-2_4](https://doi.org/10.1007/978-3-642-33090-2_4)
6. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Alternative routes in road networks. ACM J. Exp. Algorithm. **18**(1), 1–17 (2013)
7. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway dimension, shortest paths, and provably efficient algorithms. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 782–793. SIAM (2010)
8. Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.: Faster algorithms for the shortest path problem. J. ACM **37**(2), 213–223 (1990)
9. Ahuja, R.K., Orlin, J.B., Pallottino, S., Scutellà, M.G.: Dynamic shortest paths minimizing travel times and costs. Networks **41**(4), 197–205 (2003)
10. Aifadopoulou, G., Ziliaskopoulos, A., Chrisohoou, E.: Multiobjective optimum path algorithm for passenger pretrip planning in multimodal transportation networks. J. Transp. Res. Board **2032**(1), 26–34 (2007). doi:[10.3141/2032-04](https://doi.org/10.3141/2032-04)
11. Akiba, T., Iwata, Y., Kawarabayashi, K., Kawata, Y.: Fast shortest-path distance queries on road networks by pruned highway labeling. In: Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX 2014), pp. 147–154. SIAM (2014)
12. Akiba, T., Iwata, Y., Yoshida, Y.: Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013), pp. 349–360. ACM Press (2013)
13. Allulli, L., Italiano, G.F., Santaroni, F.: Exploiting GPS data in public transport journey planners. In: Gudmundsson, J., Katajainen, J. (eds.) SEA 2014. LNCS, vol. 8504, pp. 295–306. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-07959-2_25](https://doi.org/10.1007/978-3-319-07959-2_25)
14. Antsfeld, L., Walsh, T.: Finding multi-criteria optimal paths in multi-modal public transportation networks using the transit algorithm. In: Proceedings of the 19th ITS World Congress (2012)
15. Arz, J., Luxen, D., Sanders, P.: Transit node routing reconsidered. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) SEA 2013. LNCS, vol. 7933, pp. 55–66. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38527-8_7](https://doi.org/10.1007/978-3-642-38527-8_7)

16. Babenko, M., Goldberg, A.V., Gupta, A., Nagarajan, V.: Algorithms for hub label optimization. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 69–80. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-39206-1_7](https://doi.org/10.1007/978-3-642-39206-1_7)
17. Bader, R., Dees, J., Geisberger, R., Sanders, P.: Alternative route graphs in road networks. In: Marchetti-Spaccamela, A., Segal, M. (eds.) TAPAS 2011. LNCS, vol. 6595, pp. 21–32. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-19754-3_5](https://doi.org/10.1007/978-3-642-19754-3_5)
18. Barrett, C., Bisset, K., Holzer, M., Konjevod, G., Marathe, M., 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). doi:[10.1007/978-3-540-68880-8_5](https://doi.org/10.1007/978-3-540-68880-8_5)
19. Barrett, C., Bisset, K., Holzer, M., Konjevod, G., Marathe, M.V., Wagner, D.: Engineering label-constrained shortest-path algorithms. In: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74, pp. 309–319. American Mathematical Society (2009)
20. Barrett, C., Bisset, K., Jacob, R., Konjevod, G., Marathe, M.: Classical and contemporary shortest path problems in road networks: implementation and experimental analysis of the TRANSIMS router. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 126–138. Springer, Heidelberg (2002). doi:[10.1007/3-540-45749-6_15](https://doi.org/10.1007/3-540-45749-6_15)
21. Barrett, C., Jacob, R., Marathe, M.V.: Formal-language-constrained path problems. *SIAM J. Comput.* **30**(3), 809–837 (2000)
22. Bast, H.: Car or public transport—two worlds. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 355–367. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-03456-5_24](https://doi.org/10.1007/978-3-642-03456-5_24)
23. Bast, H., Brodesser, M., Storandt, S.: Result diversity for multi-modal route planning. In: Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013), OpenAccess Series in Informatics (OASICS), pp. 123–136 (2013)
24. Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., Viger, F.: Fast routing in very large public transportation networks using transfer patterns. In: Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 290–301. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-15775-2_25](https://doi.org/10.1007/978-3-642-15775-2_25)
25. Bast, H., Sternisko, J., Storandt, S.: Delay-robustness of transfer patterns in public transportation route planning. In: Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013), OpenAccess Series in Informatics (OASICS), pp. 42–54 (2013)
26. Bast, H., Storandt, S.: Flow-based guidebook routing. In: Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX 2014), pp. 155–165. SIAM (2014)
27. Bast, H., Storandt, S.: Frequency-based search for public transit. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 13–22. ACM Press, November 2014
28. Bast, H., Funke, S., Matijevic, D.: Ultrafast shortest-path queries via transit nodes. In: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74, pp. 175–192. American Mathematical Society (2009)
29. 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 (2007)

30. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. *Science* **316**(5824), 566 (2007)
31. Batz, G.V., Geisberger, R., Luxen, D., Sanders, P., Zubkov, R.: Efficient route compression for hybrid route planning. In: Even, G., Rawitz, D. (eds.) *MedAlg 2012*. LNCS, vol. 7659, pp. 93–107. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-34862-4_7](https://doi.org/10.1007/978-3-642-34862-4_7)
32. Batz, G.V., Geisberger, R., Sanders, P., Vetter, C.: Minimum time-dependent travel times with contraction hierarchies. *ACM J. Exp. Algorithm.* **18**(1.4), 1–43 (2013)
33. Batz, G.V., Sanders, P.: Time-dependent route planning with generalized objective functions. In: Epstein, L., Ferragina, P. (eds.) *ESA 2012*. LNCS, vol. 7501, pp. 169–180. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-33090-2_16](https://doi.org/10.1007/978-3-642-33090-2_16)
34. Bauer, A.: Multimodal profile queries. Bachelor thesis, Karlsruhe Institute of Technology, May 2012
35. Bauer, R., Baum, M., Rutter, I., Wagner, D.: On the complexity of partitioning graphs for arc-flags. *J. Graph Algorithms Appl.* **17**(3), 265–299 (2013)
36. Bauer, R., Columbus, T., Katz, B., Krug, M., Wagner, D.: Preprocessing speed-up techniques is hard. In: Calamoneri, T., Diaz, J. (eds.) *CIAC 2010*. LNCS, vol. 6078, pp. 359–370. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13073-1_32](https://doi.org/10.1007/978-3-642-13073-1_32)
37. Bauer, R., Columbus, T., Rutter, I., Wagner, D.: Search-space size in contraction hierarchies. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) *ICALP 2013*. LNCS, vol. 7965, pp. 93–104. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-39206-1_9](https://doi.org/10.1007/978-3-642-39206-1_9)
38. Bauer, R., D’Angelo, G., Delling, D., Schumm, A., Wagner, D.: The shortcut problem - complexity and algorithms. *J. Graph Algorithms Appl.* **16**(2), 447–481 (2012)
39. Bauer, R., Delling, D.: SHARC: Fast and robust unidirectional routing. *ACM J. Exp. Algorithm.* **14**(2.4), 1–29 (2009). Special Section on Selected Papers from *ALENEX 2008*
40. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining hierarchical, goal-directed speed-up techniques for Dijkstra’s algorithm. *ACM J. Exp. Algorithm.* **15**(2.3), 1–31 (2010). Special Section devoted to *WEA 2008*
41. Bauer, R., Delling, D., Wagner, D.: Experimental study on speed-up techniques for timetable information systems. *Networks* **57**(1), 38–52 (2011)
42. Bauer, R., Krug, M., Meinert, S., Wagner, D.: Synthetic road networks. In: Chen, B. (ed.) *AAIM 2010*. LNCS, vol. 6124, pp. 46–57. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-14355-7_6](https://doi.org/10.1007/978-3-642-14355-7_6)
43. Baum, M., Dibbelt, J., Hübschle-Schneider, L., Pajor, T., Wagner, D.: Speed-consumption tradeoff for electric vehicle route planning. In: *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2014)*, OpenAccess Series in Informatics (OASICS), pp. 138–151 (2014)
44. Baum, M., Dibbelt, J., Pajor, T., Wagner, D.: Energy-optimal routes for electric vehicles. In: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 54–63. ACM Press (2013)
45. Baumann, N., Schmidt, R.: Buxtehude-Garmisch in 6 Sekunden. die elektronische Fahrplanauskunft (EFA) der Deutschen Bundesbahn. *Zeitschrift für aktuelle Verkehrsfragen* **10**, 929–931 (1988)

46. Bellman, R.: On a routing problem. *Q. Appl. Math.* **16**, 87–90 (1958)
47. Berger, A., Dellling, D., Gebhardt, A., Müller-Hannemann, M.: Accelerating time-dependent multi-criteria timetable information is harder than expected. In: *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2009)*, OpenAccess Series in Informatics (OASICS) (2009)
48. Berger, A., Gebhardt, A., Müller-Hannemann, M., Ostrowski, M.: Stochastic delay prediction in large train networks. In: *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2011)*, OpenAccess Series in Informatics (OASICS), vol. 20, pp. 100–111 (2011)
49. Berger, A., Grimmer, M., Müller-Hannemann, M.: Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks. In: Festa, P. (ed.) *SEA 2010. LNCS*, vol. 6049, pp. 35–46. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13193-6_4](https://doi.org/10.1007/978-3-642-13193-6_4)
50. Bielli, M., Boulmakoul, A., Mouncif, H.: Object modeling and path computation for multimodal travel systems. *Eur. J. Oper. Res.* **175**(3), 1705–1730 (2006)
51. Böhmová, K., Mihalák, M., Pröger, T., Šrámek, R., Widmayer, P.: Robust routing in urban public transportation: how to find reliable journeys based on past observations. In: *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013)*, OpenAccess Series in Informatics (OASICS), pp. 27–41 (2013)
52. Botea, A.: Ultra-fast optimal pathfinding without runtime search. In: *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2011)*, pp. 122–127. AAAI Press (2011)
53. Botea, A., Harabor, D.: Path planning with compressed all-pairs shortest paths data. In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, AAAI Press (2013)
54. Brandes, U., Erlebach, T.: *Network Analysis: Methodological Foundations. Theoretical Computer Science and General Issues*, vol. 3418. Springer, Heidelberg (2005)
55. Brandes, U., Schulz, F., Wagner, D., Willhalm, T.: Travel planning with self-made maps. In: Buchsbaum, A.L., Snoeyink, J. (eds.) *ALLENEX 2001. LNCS*, vol. 2153, pp. 132–144. Springer, Heidelberg (2001). doi:[10.1007/3-540-44808-X_10](https://doi.org/10.1007/3-540-44808-X_10)
56. Brodal, G., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. In: *Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003)*, *Electronic Notes in Theoretical Computer Science*, vol. 92, pp. 3–15 (2004)
57. Bruera, F., Cicerone, S., D’Angelo, G., Di Stefano, G., Frigioni, D.: Dynamic multi-level overlay graphs for shortest paths. *Math. Comput. Sci.* **1**(4), 709–736 (2008)
58. Brunel, E., Dellling, D., Gamsa, A., Wagner, D.: Space-efficient SHARC-routing. In: Festa, P. (ed.) *SEA 2010. LNCS*, vol. 6049, pp. 47–58. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13193-6_5](https://doi.org/10.1007/978-3-642-13193-6_5)
59. Caldwell, T.: On finding minimum routes in a network with turn penalties. *Commun. ACM* **4**(2), 107–108 (1961)
60. Cambridge Vehicle Information Technology Ltd. Choice routing (2005). <http://www.camvit.com>
61. Cherkassky, B.V., Goldberg, A.V., Radzik, T.: Shortest paths algorithms. *Math. Programm. Ser. A* **73**, 129–174 (1996)

62. Cherkassky, B.V., Goldberg, A.V., Silverstein, C.: Buckets, heaps, lists, and monotone priority queues. In: *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pp. 83–92. IEEE Computer Society Press (1997)
63. Cionini, A., D'Angelo, G., D'Emidio, M., Frigioni, D., Giannakopoulou, K., Paraskevopoulos, A.: Engineering graph-based models for dynamic timetable information systems. In: *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2014)*, OpenAccess Series in Informatics (OASiCs), pp. 46–61 (2014)
64. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* **32**(5), 1338–1355 (2003)
65. Cooke, K., Halsey, E.: The shortest route through a network with time-dependent intermodal transit times. *J. Math. Anal. Appl.* **14**(3), 493–498 (1966)
66. D'Angelo, G., D'Emidio, M., Frigioni, D., Vitale, C.: Fully dynamic maintenance of arc-flags in road networks. In: Klasing, R. (ed.) *SEA 2012*. LNCS, vol. 7276, pp. 135–147. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30850-5_13](https://doi.org/10.1007/978-3-642-30850-5_13)
67. George, B.D.: *Linear Programming and Extensions*. Princeton University Press, Princeton (1962)
68. Dean, B.C.: Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
69. Dean, B.C.: Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks* **44**(1), 41–46 (2004)
70. Dean, B.C.: Shortest paths in FIFO time-dependent networks: theory and algorithms. Technical report, Massachusetts Institute Of Technology (2004)
71. Dehne, F., Omran, M.T., Sack, J.-R.: Shortest paths in time-dependent FIFO networks. *Algorithmica* **62**, 416–435 (2012)
72. Delling, D.: Time-dependent SHARC-routing. *Algorithmica* **60**(1), 60–94 (2011)
73. Delling, D., Dibbelt, J., Pajor, T., Wagner, D., Werneck, R.F.: Computing multimodal journeys in practice. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) *SEA 2013*. LNCS, vol. 7933, pp. 260–271. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38527-8_24](https://doi.org/10.1007/978-3-642-38527-8_24)
74. Delling, D., Giannakopoulou, K., Wagner, D., Zaroliagis, C.: Timetable information updating in case of delays: modeling issues. Technical report 133, Arrival Technical report (2008)
75. Delling, D., Goldberg, A.V., Nowatzyk, A., Werneck, R.F.: PHAST: Hardware-accelerated shortest path trees. *J. Parallel Distrib. Comput.* **73**(7), 940–952 (2013)
76. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Customizable route planning. In: Pardalos, P.M., Rebennack, S. (eds.) *SEA 2011*. LNCS, vol. 6630, pp. 376–387. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-20662-7_32](https://doi.org/10.1007/978-3-642-20662-7_32)
77. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Robust distance queries on massive networks. In: Schulz, A.S., Wagner, D. (eds.) *ESA 2014*. LNCS, vol. 8737, pp. 321–333. Springer, Heidelberg (2014). doi:[10.1007/978-3-662-44777-2_27](https://doi.org/10.1007/978-3-662-44777-2_27)
78. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Customizable route planning in road networks. *Transp. Sci.* (2015)
79. Delling, D., Goldberg, A.V., Razenshteyn, I., Werneck, R.F.: Graph partitioning with natural cuts. In: *25th International Parallel and Distributed Processing Symposium (IPDPS 2011)*, pp. 1135–1146. IEEE Computer Society (2011)
80. Delling, D., Goldberg, A.V., Savchenko, R., Werneck, R.F.: Hub labels: theory and practice. In: Gudmundsson, J., Katajainen, J. (eds.) *SEA 2014*. LNCS, vol. 8504, pp. 259–270. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-07959-2_22](https://doi.org/10.1007/978-3-319-07959-2_22)

81. Delling, D., Goldberg, A.V., Werneck, R.F.: Faster batched shortest paths in road networks. In: Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2011), OpenAccess Series in Informatics (OASICS), vol. 20, pp. 52–63 (2011)
82. Delling, D., Goldberg, A.V., Werneck, R.F.: Hub label compression. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) SEA 2013. LNCS, vol. 7933, pp. 18–29. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38527-8_4](https://doi.org/10.1007/978-3-642-38527-8_4)
83. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-performance multi-level routing. In: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74, pp. 73–92. American Mathematical Society (2009)
84. Delling, D., Italiano, G.F., Pajor, T., Santaroni, F.: Better transit routing by exploiting vehicle GPS data. In: Proceedings of the 7th ACM SIGSPATIAL International Workshop on Computational Transportation Science. ACM Press, November 2014
85. Delling, D., Katz, B., Pajor, T.: Parallel computation of best connections in public transportation networks. *ACM J. Exp. Algorithm.* **17**(4), 4. 1–4. 26 (2012)
86. Delling, D., Kobitzsch, M., Luxen, D., Werneck, R.F.: Robust mobile route planning with limited connectivity. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX 2012), pp. 150–159. SIAM (2012)
87. Delling, D., Kobitzsch, M., Werneck, R.F.: Customizing driving directions with GPUs. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 728–739. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-09873-9_61](https://doi.org/10.1007/978-3-319-09873-9_61)
88. Delling, D., Nannicini, G.: Core routing on dynamic time-dependent road networks. *Inform. J. Comput.* **24**(2), 187–201 (2012)
89. Delling, D., Pajor, T., Wagner, D.: Accelerating multi-modal route planning by access-nodes. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 587–598. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-04128-0_53](https://doi.org/10.1007/978-3-642-04128-0_53)
90. Delling, D., Pajor, T., Wagner, D.: Engineering time-expanded graphs for faster timetable information. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 182–206. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-05465-5_7](https://doi.org/10.1007/978-3-642-05465-5_7)
91. Delling, D., Pajor, T., Wagner, D., Zaroliagis, C.: Efficient route planning in flight networks. In: Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2009), OpenAccess Series in Informatics (OASICS) (2009)
92. Delling, D., Pajor, T., Werneck, R.F.: Round-based public transit routing. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX 2012), pp. 130–140. SIAM (2012)
93. Delling, D., Pajor, T., Werneck, R.F.: Round-based public transit routing. *Transp. Sci.* **49**, 591–604 (2014)
94. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-02094-0_7](https://doi.org/10.1007/978-3-642-02094-0_7)
95. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway hierarchies star. In: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74, pp. 141–174. American Mathematical Society (2009)

96. 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). doi:[10.1007/978-3-540-72845-0_5](https://doi.org/10.1007/978-3-540-72845-0_5)
97. Delling, D., Wagner, D.: Pareto paths with SHARC. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 125–136. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-02011-7_13](https://doi.org/10.1007/978-3-642-02011-7_13)
98. Delling, D., Wagner, D.: Time-dependent route planning. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 207–230. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-05465-5_8](https://doi.org/10.1007/978-3-642-05465-5_8)
99. Delling, D., Werneck, R.F.: Customizable point-of-interest queries in road networks. In: Proceedings of the 21st ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS 2013), pp. 490–493. ACM Press (2013)
100. Delling, D., Werneck, R.F.: Faster customization of road networks. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) SEA 2013. LNCS, vol. 7933, pp. 30–42. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38527-8_5](https://doi.org/10.1007/978-3-642-38527-8_5)
101. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74. American Mathematical Society, Providence (2009)
102. Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: A case for time-dependent shortest path computation in spatial networks. In: Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS 2010), pp. 474–477 (2010)
103. Denardo, E.V., Fox, B.L.: Shortest-route methods: 1. reaching, pruning, and buckets. *Oper. Res.* **27**(1), 161–186 (1979)
104. Dial, R.B.: Algorithm 360: shortest-path forest with topological ordering [H]. *Commun. ACM* **12**(11), 632–633 (1969)
105. Dibbelt, J., Pajor, T., Strasser, B., Wagner, D.: Intriguingly simple and fast transit routing. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) SEA 2013. LNCS, vol. 7933, pp. 43–54. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38527-8_6](https://doi.org/10.1007/978-3-642-38527-8_6)
106. Dibbelt, J., Pajor, T., Wagner, D.: User-constrained multi-modal route planning. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX 2012), pp. 118–129. SIAM (2012)
107. Dibbelt, J., Strasser, B., Wagner, D.: Customizable contraction hierarchies. In: Gudmundsson, J., Katajainen, J. (eds.) SEA 2014. LNCS, vol. 8504, pp. 271–282. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-07959-2_23](https://doi.org/10.1007/978-3-319-07959-2_23)
108. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numer. Math.* **1**, 269–271 (1959)
109. 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). doi:[10.1007/978-3-540-68552-4_26](https://doi.org/10.1007/978-3-540-68552-4_26)
110. Drews, F., Luxen, D.: Multi-hop ride sharing. In: Proceedings of the 5th International Symposium on Combinatorial Search (SoCS 2012), pp. 71–79. AAAI Press (2013)
111. Dreyfus, S.E.: An appraisal of some shortest-path algorithms. *Oper. Res.* **17**(3), 395–412 (1969)

112. Efentakis, A., Pfoser, D.: Optimizing landmark-based routing and preprocessing. In: Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science, pp. 25:25–25:30. ACM Press, November 2013
113. Efentakis, A., Pfoser, D.: GRASP. Extending graph separators for the single-source shortest-path problem. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 358–370. Springer, Heidelberg (2014). doi:[10.1007/978-3-662-44777-2_30](https://doi.org/10.1007/978-3-662-44777-2_30)
114. Efentakis, A., Pfoser, D., Vassiliou, Y.: SALT: a unified framework for all shortest-path query variants on road networks. CoRR, abs/1411.0257 (2014)
115. Efentakis, A., Pfoser, D., Voisard, A.: Efficient data management in support of shortest-path computation. In: Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science, pp. 28–33. ACM Press (2011)
116. Efentakis, A., Theodorakis, D., Pfoser, D.: Crowdsourcing computing resources for shortest-path computation. In: Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS 2012), pp. 434–437. ACM Press (2012)
117. Ehrgott, M., Gandibleux, X.: Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys. Kluwer Academic Publishers Group, New York (2002)
118. Eisenstat, D.: Random road networks: the quadtree model. In: Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2011), pp. 76–84. SIAM, January 2011
119. Eisner, J., Funke, S.: Sequenced route queries: getting things done on the way back home. In: Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS 2012), pp. 502–505. ACM Press (2012)
120. Eisner, J., Funke, S.: Transit nodes - lower bounds and refined construction. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX 2012), pp. 141–149. SIAM (2012)
121. Eisner, J., Funke, S., Herbst, A., Spillner, A., Storandt, S.: Algorithms for matching and predicting trajectories. In: Proceedings of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX 2011), pp. 84–95. SIAM (2011)
122. Eisner, J., Funke, S., Storandt, S.: Optimal route planning for electric vehicles in large network. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence. AAAI Press, August 2011
123. Eppstein, D., Goodrich, M.T.: Studying (non-planar) road networks through an algorithmic lens. In: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS 2008), pp. 1–10. ACM Press (2008)
124. Erb, S., Kobitzsch, M., Sanders, P.: Parallel bi-objective shortest paths using weight-balanced B-trees with bulk updates. In: Gudmundsson, J., Katajainen, J. (eds.) SEA 2014. LNCS, vol. 8504, pp. 111–122. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-07959-2_10](https://doi.org/10.1007/978-3-319-07959-2_10)
125. Firmani, D., Italiano, G.F., Laura, L., Santaroni, F.: Is timetabling routing always reliable for public transport? In: Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013), OpenAccess Series in Informatics (OASICS), pp. 15–26 (2013)
126. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
127. Ford, Jr., L.R.: Network flow theory. Technical report P-923, Rand Corporation, Santa Monica, California (1956)

128. Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. *Algorithmica* **68**(4), 1075–1097 (2014)
129. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **34**(3), 596–615 (1987)
130. Fu, L., Sun, D., Rilett, L.R.: Heuristic shortest path algorithms for transportation applications: state of the art. *Comput. Oper. Res.* **33**(11), 3324–3343 (2006)
131. Funke, S., Nusser, A., Storandt, S.: On k-path covers and their applications. In: *Proceedings of the 40th International Conference on Very Large Databases (VLDB 2014)*, pp. 893–902 (2014)
132. Funke, S., Nusser, A., Storandt, S.: Placement of loading stations for electric vehicles: no detours necessary! In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press (2014)
133. Funke, S., Storandt, S.: Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In: *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments (ALENEX 2013)*, pp. 31–54. SIAM (2013)
134. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. *Distrib. Comput.* **16**(2–3), 111–120 (2003)
135. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. *J. Algorithms* **53**, 85–112 (2004)
136. Geisberger, R.: Contraction of timetable networks with realistic transfers. In: Festa, P. (ed.) *SEA 2010. LNCS*, vol. 6049, pp. 71–82. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13193-6_7](https://doi.org/10.1007/978-3-642-13193-6_7)
137. Geisberger, R.: Advanced route planning in transportation networks. Ph.D. thesis, Karlsruhe Institute of Technology, February 2011
138. Geisberger, R., Kobitzsch, M., Sanders, P.: Route planning with flexible objective functions. In: *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX 2010)*, pp. 124–137. SIAM (2010)
139. Geisberger, R., Luxen, D., Sanders, P., Neubauer, S., Volker, L.: Fast detour computation for ride sharing. In: *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2010)*, OpenAccess Series in Informatics (OASICS), vol. 14, pp. 88–99 (2010)
140. Geisberger, R., Rice, M., Sanders, P., Tsotras, V.: Route planning with flexible edge restrictions. *ACM J. Exp. Algorithm.* **17**(1), 1–20 (2012)
141. Geisberger, R., Sanders, P.: Engineering time-dependent many-to-many shortest paths computation. In: *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2010)*, OpenAccess Series in Informatics (OASICS), vol. 14 (2010)
142. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. *Transp. Sci.* **46**(3), 388–404 (2012)
143. Geisberger, R., Schieferdecker, D.: Heuristic contraction hierarchies with approximation guarantee. In: *Proceedings of the 3rd International Symposium on Combinatorial Search (SoCS 2010)*. AAAI Press (2010)
144. Geisberger, R., Vetter, C.: Efficient routing in road networks with turn costs. In: Pardalos, P.M., Rebennack, S. (eds.) *SEA 2011. LNCS*, vol. 6630, pp. 100–111. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-20662-7_9](https://doi.org/10.1007/978-3-642-20662-7_9)
145. Goerigk, M., Heße, S., Müller-Hannemann, M., Schmidt, M.: Recoverable robust timetable information. In: *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013)*, OpenAccess Series in Informatics (OASICS), pp. 1–14 (2013)

146. Goerigk, M., Knott, M., Müller-Hannemann, M., Schmidt, M., Schöbel, A.: The price of strict and light robustness in timetable information. *Transp. Sci.* **48**, 225–242 (2014)
147. Goldberg, A.V.: A practical shortest path algorithm with linear expected time. *SIAM J. Comput.* **37**, 1637–1655 (2008)
148. 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. SIAM (2005)
149. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: shortest path algorithms with preprocessing. In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS Book, vol. 74, pp. 93–139. American Mathematical Society (2009)
150. 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 (2005)
151. Goldman, R., Shivakumar, N.R., Venkatasubramanian, S., Garcia-Molina, H.: Proximity search in databases. In: *Proceedings of the 24th International Conference on Very Large Databases (VLDB 1998)*, pp. 26–37. Morgan Kaufmann, August 1998
152. Goodrich, M.T., Pszona, P.: Two-phase bicriterion search for finding fast and efficient electric vehicle routes. In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Press, November 2014
153. Gunkel, T., Schnee, M., Müller-Hannemann, M.: How to find good night train connections. *Networks* **57**(1), 19–27 (2011)
154. 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 (2004)
155. Hansen, P.: Bricriteria path problems. In: Fandel, G., Gal, T. (eds.) *Multiple Criteria Decision Making - Theory and Application*. LNEMS, vol. 177, pp. 109–127. Springer, Heidelberg (1979). doi:[10.1007/978-3-642-48782-8_9](https://doi.org/10.1007/978-3-642-48782-8_9)
156. Hart, P.E., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**, 100–107 (1968)
157. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast point-to-point shortest path computations with arc-flags. In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS Book, vol. 74, pp. 41–72. American Mathematical Society (2009)
158. Hliněný, P., Moriš, O.: Scope-based route planning. In: Demetrescu, C., Halldórsson, M.M. (eds.) *ESA 2011*. LNCS, vol. 6942, pp. 445–456. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-23719-5_38](https://doi.org/10.1007/978-3-642-23719-5_38)
159. Holzer, M.: Engineering planar-separator and shortest-path algorithms. Ph.D. thesis, Karlsruhe Institute of Technology (KIT) - Department of Informatics (2008)
160. Holzer, M., Schulz, F., Wagner, D.: Engineering multilevel overlay graphs for shortest-path queries. *ACM J. Exp. Algorithm.* **13**(2.5), 1–26 (2008)
161. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining speed-up techniques for shortest-path computations. *ACM J. Exp. Algorithm.* **10**(2.5), 1–18 (2006)
162. Horvitz, E., Krumm, J.: Some help on the way: opportunistic routing under uncertainty. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (Ubicomp 2012)*, pp. 371–380. ACM Press (2012)

163. Ikeda, T., Hsu, M.-Y., Imai, H., Nishimura, S., Shimoura, H., Hashimoto, T., Tenmoku, K., Mitoh, K.: A fast algorithm for finding better routes by AI search techniques. In: Proceedings of the Vehicle Navigation and Information Systems Conference (VNSI 1994), pp. 291–296. ACM Press (1994)
164. Jing, N., Huang, Y.-W., Rundensteiner, E.A.: Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation. *IEEE Trans. Knowl. Data Eng.* **10**(3), 409–432 (1998)
165. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.* **14**(5), 1029–1046 (2002)
166. Kaindl, H., Kainz, G.: Bidirectional heuristic search reconsidered. *J. Artif. Intell. Res.* **7**, 283–317 (1997)
167. Kaufmann, H.: Towards mobile time-dependent route planning. Bachelor thesis, Karlsruhe Institute of Technology (2013)
168. Kieritz, T., Luxen, D., Sanders, P., Vetter, C.: Distributed time-dependent contraction hierarchies. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 83–93. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13193-6_8](https://doi.org/10.1007/978-3-642-13193-6_8)
169. Kirchler, D., Liberti, L., Wolfler Calvo, R.: A label correcting algorithm for the shortest path problem on a multi-modal route network. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 236–247. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30850-5_21](https://doi.org/10.1007/978-3-642-30850-5_21)
170. Kirchler, D., Liberti, L., Pajor, T., Calvo, R.W.: UniALT for regular language constraint shortest paths on a multi-modal transportation network. In: Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2011), OpenAccess Series in Informatics (OASICS), vol. 20, pp. 64–75 (2011)
171. Kleinberg, J.M., Slivkins, A., Wexler, T.: Triangulation and embedding using small sets of beacons. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), pp. 444–453. IEEE Computer Society Press (2004)
172. 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 (2007)
173. Kobitzsch, M.: HiDAR: an alternative approach to alternative routes. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 613–624. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-40450-4_52](https://doi.org/10.1007/978-3-642-40450-4_52)
174. Kobitzsch, M., Radermacher, M., Schieferdecker, D.: Evolution and evaluation of the penalty method for alternative graphs. In: Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013), OpenAccess Series in Informatics (OASICS), pp. 94–107 (2013)
175. Kontogiannis, S., Zaroliagis, C.: Distance oracles for time-dependent networks. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 713–725. Springer, Heidelberg (2014). doi:[10.1007/978-3-662-43948-7_59](https://doi.org/10.1007/978-3-662-43948-7_59)
176. Krumm, J., Gruen, R., Delling, D.: From destination prediction to route prediction. *J. Locat. Based Serv.* **7**(2), 98–120 (2013)
177. Krumm, J., Horvitz, E.: Predestination: where do you want to go today? *IEEE Comput.* **40**(4), 105–107 (2007)

178. Lauther, U.: An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, vol. 74, DIMACS Book, pp. 19–40. American Mathematical Society (2009)
179. Ken, C.K., Lee, J.L., Zheng, B., Tian, Y.: ROAD: a new spatial object search framework for road networks. *IEEE Trans. Knowl. Data Eng.* **24**(3), 547–560 (2012)
180. Lipton, R.J., Rose, D.J., Tarjan, R.: Generalized nested dissection. *SIAM J. Numer. Anal.* **16**(2), 346–358 (1979)
181. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. *SIAM J. Appl. Math.* **36**(2), 177–189 (1979)
182. Loridan, P.: ϵ -solutions in vector minimization problems. *J. Optim. Theory Appl.* **43**(2), 265–276 (1984)
183. Luxen, D., Sanders, P.: Hierarchy decomposition for faster user equilibria on road networks. In: Pardalos, P.M., Rebennack, S. (eds.) *SEA 2011. LNCS*, vol. 6630, pp. 242–253. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-20662-7_21](https://doi.org/10.1007/978-3-642-20662-7_21)
184. Luxen, D., Schieferdecker, D.: Candidate sets for alternative routes in road networks. In: Klasing, R. (ed.) *SEA 2012. LNCS*, vol. 7276, pp. 260–270. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30850-5_23](https://doi.org/10.1007/978-3-642-30850-5_23)
185. Luxen, D., Vetter, C.: Real-time routing with OpenStreetMap data. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Press (2011)
186. Madduri, K., Bader, D.A., Berry, J.W., Crobak, J.R., Parallel shortest path algorithms for solving large-scale instances. In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS Book, vol. 74, pp. 249–290. American Mathematical Society (2009)
187. Martins, E.Q.: On a multicriteria shortest path problem. *Eur. J. Oper. Res.* **26**(3), 236–245 (1984)
188. Maue, J., Sanders, P., Matijevic, D.: Goal-directed shortest-path queries using precomputed cluster distances. *ACM J. Exp. Algorithm.* **14**, 3.2:1–3.2:27 (2009)
189. Mehlhorn, K.: A faster approximation algorithm for the Steiner problem in graphs. *Inf. Process. Lett.* **27**(3), 125–128 (1988)
190. Mehlhorn, K., Sanders, P., Algorithms, D.S.: *The Basic Toolbox*. Springer, Heidelberg (2008)
191. Mellouli, T., Suhl, L.: Passenger online routing in dynamic networks. In: Mattfeld, D.C., Suhl, L. (eds.) *Informations probleme in Transport und Verkehr*, vol. 4, pp. 17–30. DS&OR Lab, Universität Paderborn (2006)
192. 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)
193. Meyer, U., Sanders, P.: δ -stepping: a parallelizable shortest path algorithm. *J. Algorithms* **49**(1), 114–152 (2003)
194. Milosavljević, N.: On optimal preprocessing for contraction hierarchies. In: *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pp. 33–38. ACM Press (2012)
195. Modesti, P., Sciomachen, A.: A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *Eur. J. Oper. Res.* **111**(3), 495–508 (1998)

196. Möhring, R.H.: Verteilte Verbindungssuche im öffentlichen Personenverkehr - Graphentheoretische Modelle und Algorithmen. In: *Angewandte Mathematik insbesondere Informatik, Beispiele erfolgreicher Wege zwischen Mathematik und Informatik*, pp. 192–220. Vieweg (1999)
197. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speedup Dijkstra's algorithm. *ACM J. Exp. Algorithm.* **11**(28), 1–29 (2006)
198. Moore, E.F.: The shortest path through a maze. In: *Proceedings of the International Symposium on the Theory of Switching*, pp. 285–292. Harvard University Press (1959)
199. Müller-Hannemann, M., Schnee, M.: Paying less for train connections with MOTIS. In: *Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2005)*, OpenAccess Series in Informatics (OASICS), p. 657 (2006)
200. Müller-Hannemann, M., Schnee, M.: Finding all attractive train connections by multi-criteria pareto search. In: Geraets, F., Kroon, L., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) *Algorithmic Methods for Railway Optimization*. LNCS, vol. 4359, pp. 246–263. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-74247-0_13](https://doi.org/10.1007/978-3-540-74247-0_13)
201. Müller-Hannemann, M., Schnee, M.: Efficient timetable information in the presence of delays. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) *Robust and Online Large-Scale Optimization*. LNCS, vol. 5868, pp. 249–272. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-05465-5_10](https://doi.org/10.1007/978-3-642-05465-5_10)
202. Müller-Hannemann, M., Schnee, M., Weihe, K.: Getting train timetables into the main storage. *Electron. Notes Theoret. Comput. Sci.* **66**(6), 8–17 (2002)
203. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: models and algorithms. In: Geraets, F., Kroon, L., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) *Algorithmic Methods for Railway Optimization*. LNCS, vol. 4359, pp. 67–90. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-74247-0_3](https://doi.org/10.1007/978-3-540-74247-0_3)
204. Müller-Hannemann, M., Weihe, K.: Pareto shortest paths is often feasible in practice. In: Brodal, G.S., Frigioni, D., Marchetti-Spaccamela, A. (eds.) *WAE 2001*. LNCS, vol. 2141, pp. 185–197. Springer, Heidelberg (2001). doi:[10.1007/3-540-44688-5_15](https://doi.org/10.1007/3-540-44688-5_15)
205. 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). doi:[10.1007/978-3-540-75520-3_58](https://doi.org/10.1007/978-3-540-75520-3_58)
206. Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. *Eur. J. Oper. Res.* **83**(1), 154–166 (1995)
207. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* search on time-dependent road networks. *Networks* **59**, 240–251 (2012). Best Paper Award
208. Orda, A., Rom, R.: Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *J. ACM* **37**(3), 607–625 (1990)
209. Orda, A., Rom, R.: Minimum weight paths in time-dependent networks. *Networks* **21**, 295–319 (1991)
210. Pajor, T.: Multi-modal route planning. Master's thesis, Universität Karlsruhe (TH), March 2009
211. Pallottino, S., Scutellà, M.G.: Shortest path algorithms in transportation models: Classical and innovative aspects. In: *Equilibrium and Advanced Transportation Modelling*, pp. 245–281. Kluwer Academic Publishers Group (1998)

212. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000), pp. 86–92 (2000)
213. Paraskevopoulos, A., Zaroliagis, C.: Improved alternative route planning. In: Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2013), OpenAccess Series in Informatics (OASICS), pp. 108–122 (2013)
214. Parter, S.V.: The use of linear graphs in Gauss elimination. *SIAM Rev.* **3**(2), 119–130 (1961)
215. Peleg, D.: Proximity-preserving labeling schemes. *J. Graph Theory* **33**(3), 167–176 (2000)
216. Pohl, I.: Bi-directional and heuristic search in path problems. Technical report SLAC-104, Stanford Linear Accelerator Center, Stanford, California (1969)
217. Pohl, I.: Bi-directional search. In: Proceedings of the Sixth Annual Machine Intelligence Workshop, vol. 6, pp. 124–140. Edinburgh University Press (1971)
218. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Experimental comparison of shortest path approaches for timetable information. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 88–99. SIAM (2004)
219. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. *ACM J. Exp. Algorithm.* **12**(24), 1–39 (2008)
220. Rice, M., Tsotras, V.: Bidirectional A* search with additive approximation bounds. In: Proceedings of the 5th International Symposium on Combinatorial Search (SoCS 2012), AAAI Press (2012)
221. Rice, M.N., Tsotras, V.J.: Exact graph search algorithms for generalized traveling salesman path problems. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 344–355. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30850-5_30](https://doi.org/10.1007/978-3-642-30850-5_30)
222. Sanders, P., Mandow, L.: Parallel label-setting multi-objective shortest path search. In: 27th International Parallel and Distributed Processing Symposium (IPDPS 2013), pp. 215–224. IEEE Computer Society (2013)
223. 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). doi:[10.1007/11561071_51](https://doi.org/10.1007/11561071_51)
224. Sanders, P., Schultes, D.: Robust, almost constant time shortest-path queries in road networks. In: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book, vol. 74, pp. 193–218. American Mathematical Society (2009)
225. Sanders, P., Schultes, D.: Engineering highway hierarchies. *ACM J. Exp. Algorithm.* **17**(1), 1–40 (2012)
226. 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). doi:[10.1007/978-3-540-87744-8_61](https://doi.org/10.1007/978-3-540-87744-8_61)
227. Sanders, P., Schulz, C.: Distributed evolutionary graph partitioning. In: Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX 2012), pp. 16–29. SIAM (2012)
228. Sankaranarayanan, J., Alborzi, H., Samet, H.: Efficient query processing on spatial networks. In: Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS 2005), pp. 200–209 (2005)
229. Sankaranarayanan, J., Samet, H.: Query processing using distance oracles for spatial networks. *IEEE Trans. Knowl. Data Eng.* **22**(8), 1158–1175 (2010)

- 230. Sankaranarayanan, J., Samet, H.: Roads belong in databases. *IEEE Data Eng. Bull.* **33**(2), 4–11 (2010)
- 231. Schilling, H.: TomTom navigation - How mathematics help getting through traffic faster (2012). Talk given at ISMP
- 232. Schreiber, R.: A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Softw.* **8**(3), 256–276 (1982)
- 233. Schultes, D.: Route planning in road networks. Ph.D. thesis, Universität Karlsruhe (TH), February 2008
- 234. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: Demetrescu, C. (ed.) *WEA 2007. LNCS*, vol. 4525, pp. 66–79. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-72845-0_6](https://doi.org/10.1007/978-3-540-72845-0_6)
- 235. Schulz, F., Wagner, D., Weihe, K.: Dijkstra’s algorithm on-line: an empirical case study from public railroad transport. *ACM J. Exp. Algorithm.* **5**(12), 1–23 (2000)
- 236. 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). doi:[10.1007/3-540-45643-0_4](https://doi.org/10.1007/3-540-45643-0_4)
- 237. Sedgewick, R., Vitter, J.S.: Shortest paths in Euclidean graphs. *Algorithmica* **1**(1), 31–48 (1986)
- 238. Sommer, C.: Shortest-path queries in static networks. *ACM Comput. Surv.* **46**(4), 1–31 (2014)
- 239. Storandt, S.: Route planning for bicycles - exact constrained shortest paths made practical via contraction hierarchy. In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, pp. 234–242 (2012)
- 240. Storandt, S., Funke, S.: Cruising with a battery-powered vehicle and not getting stranded. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press (2012)
- 241. Storandt, S., Funke, S.: Enabling e-mobility: facility location for battery loading stations. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press (2013)
- 242. Strasser, B., Wagner, D.: Connection scan accelerated. In: *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX 2014)*, pp. 125–137. SIAM (2014)
- 243. Theune, D.: Robuste und effiziente Methoden zur Lösung von Wegproblemen. Ph.D. thesis, Universität Paderborn (1995)
- 244. Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. In: *35th ACM Symposium on Theory of Computing*, pp. 149–158. ACM, New York (2003)
- 245. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM* **51**(6), 993–1024 (2004)
- 246. Tsaggouris, G., Zaroliagis, C.: Multiobjective optimization: improved FPTAS for shortest paths and non-linear objectives with applications. *Theory Comput. Syst.* **45**(1), 162–186 (2009)
- 247. Tulp, E., Siklóssy, L.: TRAINS, an active time-table searcher. *ECAI* **88**, 170–175 (1988)
- 248. Tulp, E., Siklóssy, L.: Searching time-table networks. *Artif. Intell. Eng. Des. Anal. Manuf.* **5**(3), 189–198 (1991)
- 249. van Vliet, D.: Improved shortest path algorithms for transport networks. *Transp. Res. Part B: Methodol.* **12**(1), 7–20 (1978)

250. Wagner, D., Willhalm, T.: Drawing graphs to speed up shortest-path computations. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX 2005), pp. 15–24. SIAM (2005)
251. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric containers for efficient shortest-path computation. *ACM J. Exp. Algorithm.* **10**(1.3), 1–30 (2005)
252. Weller, M.: Optimal hub labeling is NP-complete. *CoRR*, abs/1407.8373 (2014)
253. White, D.J.: Epsilon efficiency. *J. Optim. Theory Appl.* **49**(2), 319–337 (1986)
254. Williams, J.W.J.: Algorithm 232: heapsort. *J. ACM* **7**(6), 347–348 (1964)
255. Winter, S.: Modeling costs of turns in route planning. *GeoInformatica* **6**(4), 345–361 (2002)
256. Witt, S.: Trip-based public transit routing. In: Bansal, N., Finocchi, I. (eds.) *ESA 2015*. LNCS, vol. 9294, pp. 1025–1036. Springer, Heidelberg (2015). doi:[10.1007/978-3-662-48350-3_85](https://doi.org/10.1007/978-3-662-48350-3_85)
257. Lingkun, W., Xiao, X., Deng, D., Cong, G., Zhu, A.D., Zhou, S.: Shortest path and distance queries on road networks: an experimental evaluation. *Proc. VLDB Endow.* **5**(5), 406–417 (2012)
258. Yu, H., Lu, F.: Advanced multi-modal routing approach for pedestrians. In: 2nd International Conference on Consumer Electronics, Communications and Networks, pp. 2349–2352 (2012)
259. Zadeh, L.A.: Fuzzy logic. *IEEE Comput.* **21**(4), 83–93 (1988)
260. Zhong, R., Li, G., Tan, K.-L., Zhou, L.: G-tree: an efficient index for KNN search on road networks. In: Proceedings of the 22nd International Conference on Information and Knowledge Management, pp. 39–48. ACM Press (2013)
261. Zhu, A.D., Ma, H., Xiao, X., Luo, S., Tang, Y., Zhou, S.: Shortest path, distance queries on road networks: towards bridging theory and practice. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013), pp. 857–868. ACM Press (2013)
262. Zwick, U.: Exact and approximate distances in graphs — a survey. In: Heide, F.M. (ed.) *ESA 2001*. LNCS, vol. 2161, pp. 33–48. Springer, Heidelberg (2001). doi:[10.1007/3-540-44676-1_3](https://doi.org/10.1007/3-540-44676-1_3)

Algorithm Engineering

Selected Results and Surveys

Kliemann, L.; Sanders, P. (Eds.)

2016, X, 419 p. 68 illus., Softcover

ISBN: 978-3-319-49486-9