Route Planning in Transportation Networks

HANNAH BAST
University of Freiburg

DANIEL DELLING
Microsoft Research

ANDREW V. GOLDBERG
Microsoft Research

MATTHIAS MÜLLER-HANNEMANN
Martin-Luther-Universität Halle-Wittenberg

THOMAS PAJOR
Microsoft Research

PETER SANDERS
Karlsruhe Institute of Technology

DOROTHEA WAGNER
Karlsruhe Institute of Technology

RENATO F. WERNECK
Microsoft Research

January 8, 2014
Technical Report
MSR-TR-2014-4

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 approximations or simplifications. 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.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

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. [80] has appeared fairly recently, it has become outdated due to significant developments in the last half-decade. For example, for continent-sized 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).

With that in mind, 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, query times, 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 on 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 (Section 6) will explain, several algorithms we discuss have found their way into mainsteam production systems serving millions of users on a daily basis.

This survey considers research published until January 2014. 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 or 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, some 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 Section 3.

2.1 Basic Techniques

The standard solution to the one-to-all shortest path problem is Dijkstra's algorithm [91]. 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: 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 updates it 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, 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 target t. We refer to the set of vertices $S \subseteq V$ scanned by the algorithm as its *search space*. See Figure 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 [226], improving to $\mathcal{O}(|A| + |V| \log |V|)$ with Fibonacci heaps [108]. For arbitrary (non-integral) costs, generalized versions of binary heaps (such as 4-heaps or 8-heaps) tend to work best in practice [56]. If all arc costs are integers in the range $[0, C]$, multi-level buckets [88] yield a running time of $\mathcal{O}(|A|+|V| \sqrt{\log C})$ [8,57] and work well in practice. For the average case, one can get an $\mathcal{O}(|V|+|A|)$ (linear) time bound [124,166]. Thorup [217] has improved

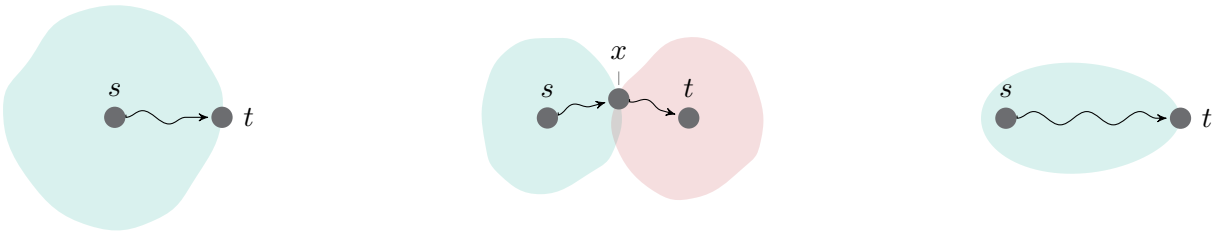

Figure 1: Schematic search spaces of Dijkstra's algorithm (left), bidirectional search (middle), and the A* algorithm (right).

the theoretical worst-case bound of Dijkstra's algorithm to $O(|A| + |V| \log \log \min\{|V|, C\})$, 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* [61], 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 [42,107,172]. 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 [106] 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 [132]. It uses a potential function $\pi\colon V \to \mathbb{R}$ 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 Figure 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 $\ell(v,w) - \pi(v) + \pi(w) \geq 0$), 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 [139]. Another approach is to change the stopping criteria instead of making the two functions consistent [125, 142, 190], but this is more complicated and performs no better in practice.

In road networks with travel time metric, one can use the geographical distance [191, 210] between u and t divided be 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 [125]. 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 on 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 [125]. 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, both $\text{dist}(u, t) \geq \text{dist}(u, l) - \text{dist}(t, l)$ and $\text{dist}(u, t) \geq \text{dist}(l, t) - \text{dist}(l, u)$ hold. The corresponding potential function is feasible [125].

The quality of the lower bounds (and thus query performance) depends on which vertices are chosen as landmarks during preprocessing. On road networks, picking well-spaced landmarks close to the boundary of the graph leads to the best results, with acceptable query times on average [95, 127]. For a small (but noticeable) fraction of the queries, however, speedups relative to bidirectional Dijkstra are minor.

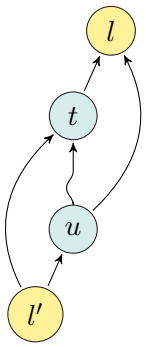

Figure 2: Triangle inequalities for ALT.

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. [208] approximate the set V_a by an angular sector (centered at u) that covers all vertices in V_a. Wagner et al. [224] 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 [50, 223]. 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 [133, 152] 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 [171]. 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 [133].

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 [133]. To reduce preprocessing space, one can use a compression scheme that flips some flags from zero to one [53], which preserves correctness. As Section 3 will show, Arc Flags currently have the fastest query times among purely goal-directed methods on road networks. Although high preprocessing times (of several hours) have long been a drawback of Arc Flags, the recent PHAST algorithm (cf. Section 2.7) can make this method more competitive with other techniques [65].

Precomputed Cluster Distances. Another goal-directed technique is *Precomputed Cluster Distances* (PCD) [162]. Like Arc Flags, it is based on a (preferably balanced) partition $\mathcal{C} = (C_1, \ldots, C_K)$ with K cells (or clusters). The preprocessing algorithm computes the shortest path distances between all pairs of cells. 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 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) [48, 49] method implicitly stores all-pairs shortest path information so that shortest paths can be quickly retrieved during queries. Each vertex $u \in G$ 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 [49] 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, run-length encoding, and sliding window compression. 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 SILC [201], and both can be seen as stronger versions of Geometric Containers.

2.3 Separator-Based Techniques

Planar graphs have small (and efficiently-computable) small separators [155]. Although road networks are not planar (think of tunnels or overpasses) they have been observed to have small separators as well [68, 104, 200]. 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 [222] 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. [208] 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 [209] to multilevel separator 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 Figure 3 for an illustration.

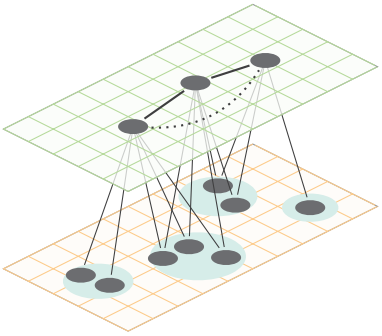

Figure 3: Multilevel overlay graph with two levels. The dots depict separator vertices in the lower (orange) and upper (green) level.

Other variants of this approach offer different trade-offs by adding many more shortcuts to the graph during preprocessing, sometimes across different levels [128, 140]. In particular *High-Performance Multilevel Routing* (HPML) [71] 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 [218] and implemented by Muller and Zachariasen [179]. 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 $\mathcal{C} = (C_1, \ldots, C_k)$ 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.

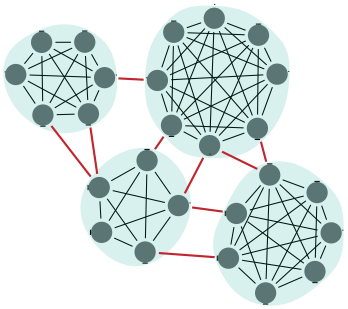

Figure 4: Overlay graph constructed from arc separators. Each cell contains a full clique between its boundary vertices, and cut arcs are thick red.

An early version of this approach is the *Hierarchical MulTi* (HiTi) method [141]. 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 [141], leading to modest speedups.

The recent *Customizable Route Planning* (CRP) [66,67] algorithm uses a similar approach, but is specifically engineered to meet the requirements of a 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 [68], 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 [86] (cf. Section 2.4), at the cost of increased space usage. 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 [96,134], though there is no guarantee that it will find exact shortest paths. Fu et al. [109] give an overview of early approaches using this technique. The algorithms we discuss in this section find exact shortest paths, and 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. [119], 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.

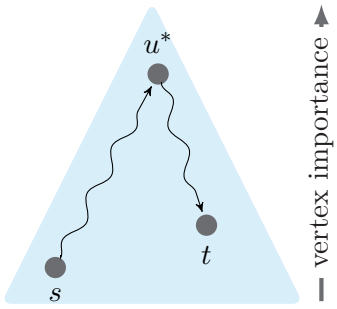

Figure 5: Illustrating a Contraction Hierarchies query.

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 Figure 5 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 pruning. Nevertheless, Geisberger et al. [119] 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, which 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 [119, 143]. 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]. Approximate CH has been considered as a way to accommodate networks with less inherent hierarchy [120].

CH is actually a successor of Highway Hierarchies [198] and Highway Node Routing [207], 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, 13, 37, 86], but also for extended queries (cf. Section 2.7) and applications (cf. Section 3.2).

Reach. Another hierarchical approach is *Reach* [130]. 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.

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 [126] 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 [130] has shown that such bounds can be obtained faster by computing partial shortest path trees. Goldberg et al. [126] 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. CH is even faster, however, and its preprocessing algorithm is much simpler.

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 [65] 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* [189]. 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) [58, 112], 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 Figure 6 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 has a hub in $L_f(s) \cap L_b(t)$.

Although the required average label size can be $\Theta(|V|)$ in general [112], 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). Even faster approaches to convert a vertex ordering into labels have subsequently been proposed by Abraham et al. [5] and Akiba et al. [11].

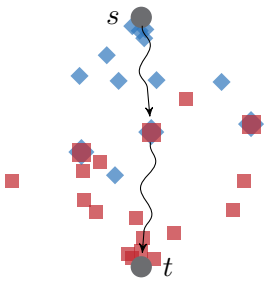

Figure 6: Illustrating hub labels of vertices s (diamonds) and t (squares).

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 Section 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) [70] reduces space usage by an order of magnitude, at the expense of higher query times.

Transit Node Routing. The *Transit Node Routing* (TNR) [13, 25, 27, 197] 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. 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 Figure 7 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 [25, 197].

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 [25]. (Although an optimized preprocessing routine for this grid-based

approach was later shown to have a flaw that could potentially result in suboptimal queries [228], the slower version reported in [25] is correct and achieves the same query times.)

For better performance [3,13,119,197], one can pick as transit nodes vertices that are classified as important by a hierarchical speedup technique (such as CH). Locality filters are less straightforward in such cases: although one can still use geographical distances [119, 197], a graph-based approach considering the Voronoi regions [163] induced by transit nodes tends to be significantly more accurate [13]. 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.

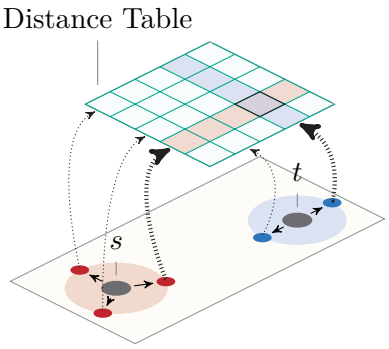

Figure 7: Illustrating a TNR query. The red (blue) dots are the access nodes of s (t). The arrows point to the respective rows/columns of the distance table. The highlighted entries correspond to the access nodes which minimize the combined s–t distance.

Pruned Highway Labeling. The Pruned Highway Labeling (PHL) [10] algorithm can be seen as a hybrid between pure labeling and transit nodes. Its preprocessing routine separates the input by 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 [11]. Although this method has some similarity with Thorup's distance oracle for planar graphs [218], 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 [137, 208] 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 [126]. 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* [37, 74]. 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 [81] is similar, but uses Highway Hierarchies [198] to determine the core.

Another approach with two-phase queries is *ReachFlags* [37]. 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 [36] 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 [53]. 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 Section 2.7).

Combining CH with Arc Flags results in the *CHASE* algorithm [37]. 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 a small fraction (about 5 %) of the total number $|V|$ of vertices [37]. More recently, Delling et al. showed that PHAST (see Section 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 [65].

Finally, Bauer et al. [37] 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.

2.7.1 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 [119] 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 [198] or run a local (bidirectional) Dijkstra search from its endpoints [67]. These two techniques can be used for bounded-hop algorithms as well.

2.7.2 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 [147], but can be used with any other hierarchical speedup technique (such as CH) and even with hub labels [69]. 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 [85]. 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 [124]. However, Dijkstra's algorithm has bad locality and is hard to parallelize, especially for sparse graphs [160, 167]. PHAST [65] 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. On 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 [69].

2.7.3 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, three 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 [224], ALT [82], Arc Flags [60], and CH [119, 207], 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 [82]. Less trivially, CH queries can also be modified to deal with dynamic changes to the network [119, 207] 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 split 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 [82, 95]. For CH, one can keep the ordering, and just rerun contraction [119, 231]. For CRP, one can keep the partitioning and the overlay topology, and just recompute the shortcut lengths [67].

2.7.4 Time-Dependence

In real transportation networks, the best route often depends on the departure time in a predictable way. 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 [59] as long as later departures cannot lead to earlier arrivals; this *non-overtaking* property is often called first-in first-out (FIFO). Many of the techniques described so far work in this scenario, including bidirectional ALT [74, 181], CH [29], or SHARC [63].

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 [29, 181]. 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 [53] or CH [29]. Batched shortest paths can be computed in such networks efficiently as well [118].

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 *profile searches*, which compute the full travel time function between two points. Most speedup techniques can be adapted to deal with this as well [29, 63].

Unfortunately, even a slight departure 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 [30]. However, a heuristic adaptation of time-dependent CH shows negligible errors in practice [30].

2.7.5 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 [117], 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. Dijkstra's algorithm can be extended to compute Pareto sets and is fast enough as long as these sets are small [177]. However, for many criteria common for transportation networks, these sets can become rather large, which makes it hard to achieve large speedups [43, 83].

A reasonable alternative [115] 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 [110] 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. [33] 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), and Contraction Hierarchies (with respect to the vertex order). In fact, minimizing the number of shortcuts for CH is APX-hard [33, 168]. For SHARC, however, a greedy factor-k approximation algorithm exists [35]. Deciding which k shortcuts (for fixed k) to add to a graph in order to minimize the SHARC search space is also NP-hard [35]. Bauer et al. [32] 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 [34, 168] have independently observed a natural relationship between CH and the notions of filled graphs [188] and elimination trees [205]. For planar graphs, one can use nested dissection [154] to build a CH order leading to $\mathcal{O}(|V| \log |V|)$ shortcuts [34, 168].

More generally, for minor-closed graph classes with balanced $\mathcal{O}(\sqrt{|V|})$-separators, the search space is bounded by $\mathcal{O}(\sqrt{|V|})$ [34]. Similarly, on graphs with treewidth k, the search space of CH is bounded by $\mathcal{O}(k \log |V|)$ [34].

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 $\mathcal{O}(\sqrt{|V|})$ [154, 155]. Although road networks are not strictly planar, they do have small separators [68, 104], so theoretically efficient algorithms for planar graphs are likely to also perform well on road networks. Sommer [211] 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 [27], 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. (Their analysis assumes 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. [146] 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 on 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 perform well 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. [39] 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 [104] and by Eisenstat [99].

Besides these results, Rice and Tsotras [194] analyze (a heuristic variant of) 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 [52]. For Transit Node Routing, Eisner and Funke [101] 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 Gavoille et al. [112]. Approximation algorithms to compute small labels have also been studied [14, 58].

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 [211], Zwick [234] and Gavoille and Peleg [111].

3 Route Planning in Road Networks

In this section, we evaluate how the techniques discussed so far perform on 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 [164], 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 [206]: 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 [227]. There are two common approaches to deal with turns. The *arc-based representation* [54] blows up the graph so that roads become vertices and feasible turns become arcs. In contrast, the *compact representation* [66, 121] 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 turn modeling than others, however, it is hard to extrapolate these results to more realistic networks. We therefore consider experimental results for each model separately.

3.1.1 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. 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 [87].

Figure 8 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

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

		DATA STRUCTURES		QUERIES	
algorithm	source	space [GiB]	time [h:m]	scanned vertices	time [µs]
Dijkstra	[65]	0.4	–	9 300 000	2 550 000
Bidir. Dijkstra	[65]	0.4	–	4 800 000	1 350 000
CRP	[67]	0.9	1:00	2 766	1 650
Arc Flags	[65]	0.6	0:20	2 646	408
CH	[67]	0.4	0:05	280	110
CHASE	[65]	0.6	0:30	28	5.76
HLC	[70]	1.8	0:50	–	2.55
TNR	[13]	2.5	0:20	–	1.25
TNR+AF	[37]	5.4	1:45	–	0.99
HL	[70]	18.8	0:37	–	0.56
HL-∞	[5]	17.7	60:00	–	0.25
table lookup	[65]	1 208 358.7	145:30	–	0.06

third important quality measure) is not explicitly represented.[1] We reproduce the numbers reported by Bauer et al. [37] 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 [65]. We also consider the recent ALT implementation by Efentakis and Pfoser [95]. We also report results for several variants of TNR [13, 37], Hub Labels [5, 70], HPML [71], and Contraction Hierarchies (CH) [119]. CRP (and the corresponding PUNCH) figures [67] use a more realistic graph model that includes turn costs. For reference, the plot includes two implementations [37] of Dijkstra's algorithm: unidirectional and bidirectional. Finally, the table-lookup figure is based on the time of a single memory access and the precomputation time of $|V|$ shortest path trees using PHAST [65]. 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, 65, 67, 70]; for the remaining, we apply scaling factors previously used in the literature: 1.915 for [37, 71] and 1.275 for [13, 119]. We use the same methodology to determine a scaling factor of 0.734 for [95].

The figure shows that there is no best technique for this instance. 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 has additional details about the methods in the Pareto set, including two versions of Dijkstra's algorithm, two Dijkstra-based hierarchical techniques (CRP and CH), three non-

[1] The reader is referred to Sommer [211] for a similar plot (which inspired ours) relating query times to preprocessing space.

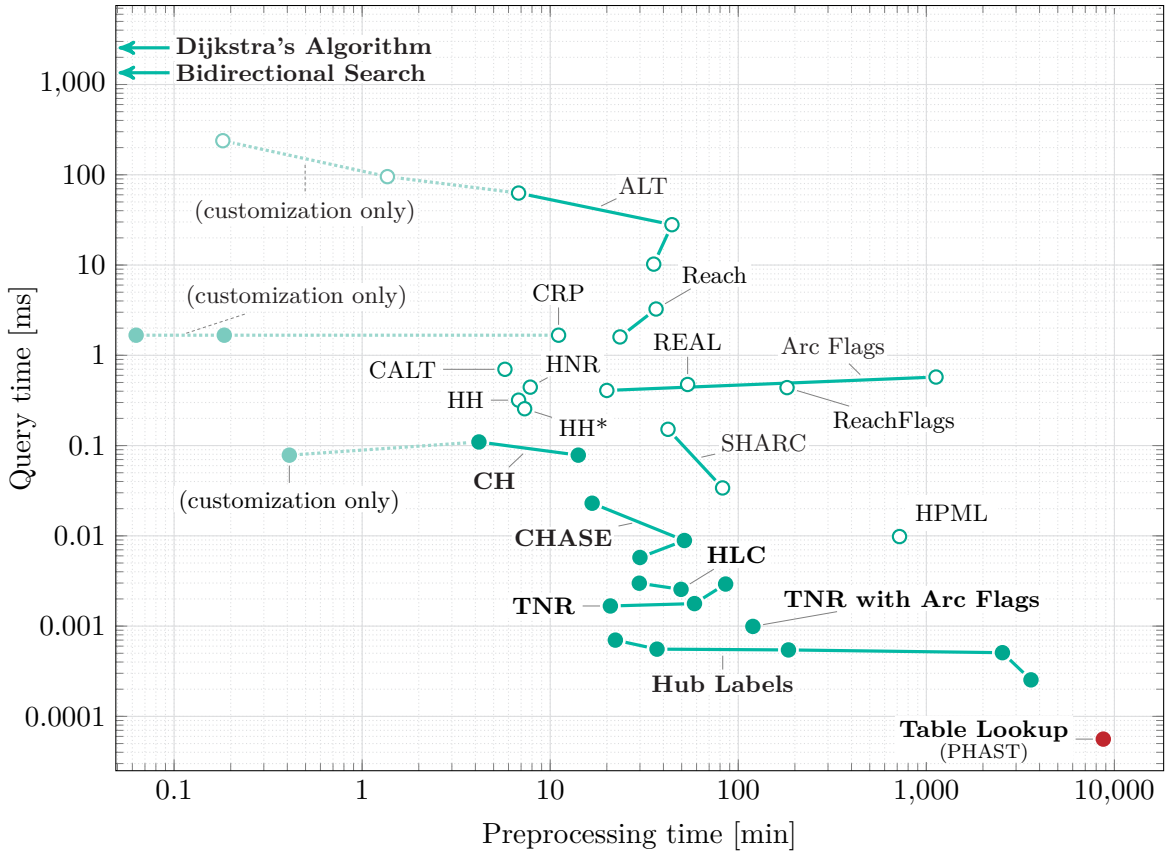

Figure 8: Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [211].

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) [65, 121].

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 Section 2 can answer

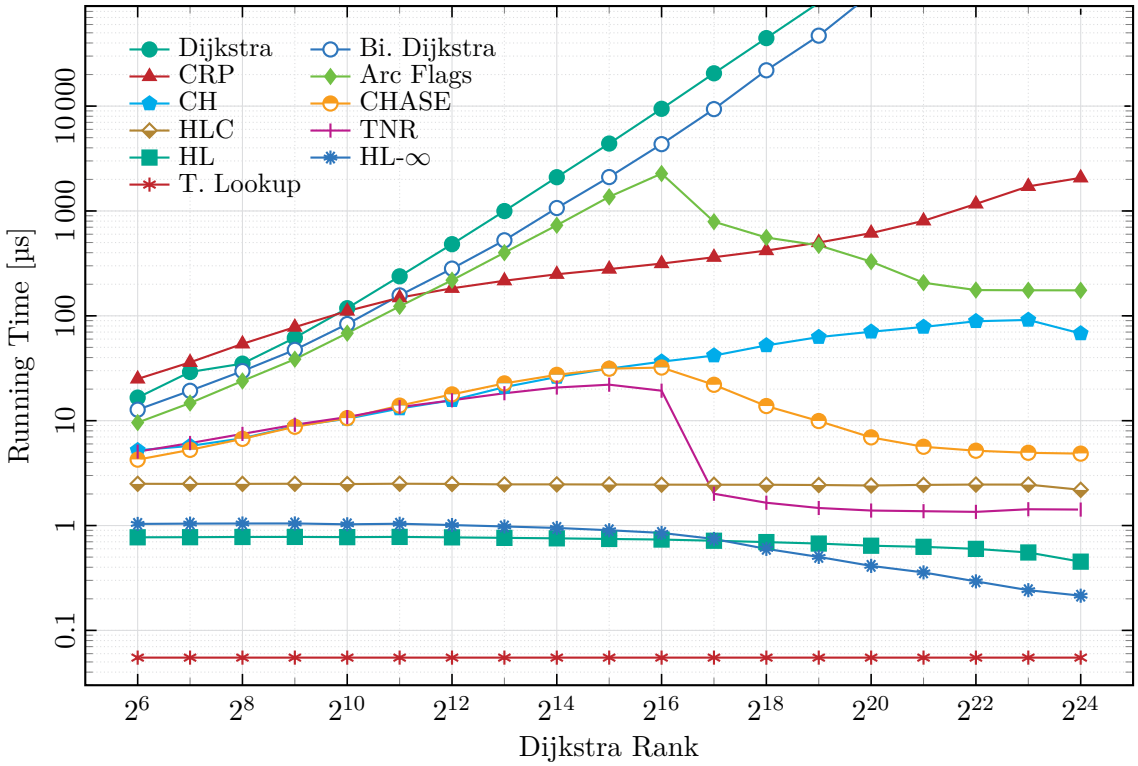

Figure 9: Performance of speedup techniques for various Dijkstra ranks.

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 consist 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 [196] 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 $2^1, 2^2, \ldots, 2^{\lfloor \log |V| \rfloor}$ for some randomly chosen sources, all types (local, mid-range, global) of queries are evaluated. Figure 9 reports the median running times for all techniques from Table 1 (except TNR+AF, for which such numbers have never been published) for 1 000 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.

3.1.2 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 under realistic constraints [67], however. In contrast, CRP was developed with these constraints in mind. As explained in Section 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 [67]) 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 [67]. 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 or the presence of turns. Also, a new cost function can be applied in roughly a second, fast enough to even support user-specific cost functions. Contraction-based customization could further reduce customization time to less than 0.4 s, though space usage would increase [67]. In contrast, CH performance is

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.

		CH					CRP						
		DS		QUERIES			DS		CUSTOM		QUERIES		
metric	turn info	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	0:11	0.82	1.04	0.07	2942	1.91	2.49
time	none	0:02	0.60	280	0.11	0.21	0:11	0.82	1.05	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	0:11	0.86	1.10	0.07	3049	1.67	1.85

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 [70]. For more dynamic scenarios, CH preprocessing can be made parallel [121] or even distributed [143]; even if run sequentially, it is fast enough for large metropolitan areas.

3.2 Applications

As discussed in Section 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) [55]. In general, these alternative paths should be short, smooth, and significatly different from the shortest path (and other alternatives). Such paths can either be computed directly as the concatenation of partial shortest paths paths [6, 55, 67, 148, 158] or compactly represented as a small graph [15, 149, 187]. A related problem is to compute a *corridor* [73] 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 [28].

Extensions that deal with nontrivial cost functions have also been considered. In particular, one can extend CH to handle flexible arc restrictions [117] (such as height or weight limitations) or even multiple criteria [110, 115] (such as optimizing costs and travel time). Minimizing the energy consumption of electric vehicles [40, 103, 213, 214] 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 [212].

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 [100, 102]. Traffic simulations also benefit from acceleration techniques [157], since they must consider the likely routes taken by *all* drivers in a network. Another application is route prediction [151]: 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, 102, 138, 150]. Another important application is *ride sharing* [2, 93, 116], 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, 85, 100, 114, 153, 195, 232]. 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. Section 2.7.2), 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, 85].

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 auxilary 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 [127] and CH [199]. 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 [203]. 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 [202], 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 [143] and CRP [97].

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.

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. 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 [41, 208, 220, 221]. 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. [176].

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) [185] 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 [170, 208] contains a vertex for every departure and arrival event, with consecutive departure and arrival events connected by *connection arcs*. To enable transfers between vehicles, all vertices at the same stop are (linearly, in chronological order) interlinked by *transfer arcs*. Müller-Hannemann and Weihe [177] 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. [192, 193] and Müller-Hannemann and Schnee [174] 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 so-called *realistic* model introduces an additional *transfer vertex* per departure event, and connects each arrival vertex to the first transfer vertex that obeys the minimum change time constraints. See Figure 10 for an illustration. This model has been further engineered [76] to reduce the number of arcs that are explored "redundantly" during queries.

Time-Dependent Model. The main disadvantage of the time-expanded model is that the resulting graphs are quite large [192]. The *time-dependent approach*, in contrast, produces significantly smaller (in terms of number of vertices and arcs) graphs by not unrolling the timetable. Instead, time dependencies are encoded by *travel time functions* on the arcs, which map departure times to travel times. Evaluating the cost of an arc then depends on the time at which it is traversed. A general analysis of time-dependent shortest paths under various waiting constraints is conducted by Orda and Rom [182, 183]. It turns out that the shortest-path problem can be efficiently solved if travel time functions are nonnegative and FIFO (which implies that waiting never pays off).

For the public transit scenario, the time-dependent approach has been considered by Brodal and Jacob [51]. 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

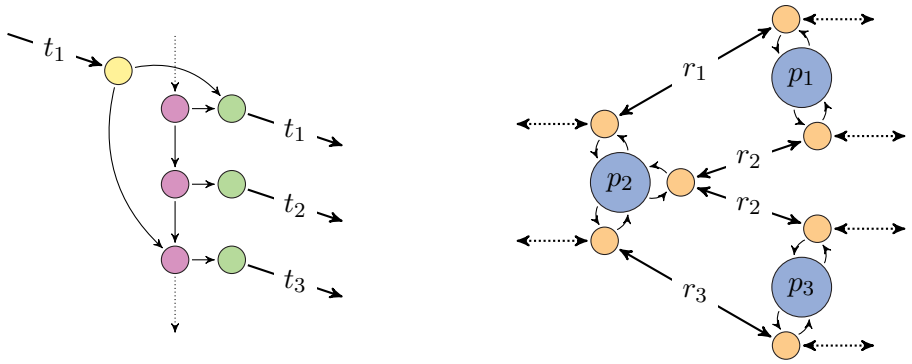

Figure 10: Realistic time-expanded (left) and time-dependent (right) models. Connection arcs in the time-expanded model are annotated with its trips t_i, and route arcs in the time-dependent model with its routes r_i.

departure and arrival times are encoded by the travel time function associated with the arc (u, v); see also Figure 11. Pyrga et al. [193] 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 Figure 10. They also consider a model that allows arbitrary minimum change times between pairs of routes within each stop [193]. 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 [72].

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* [180], 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.

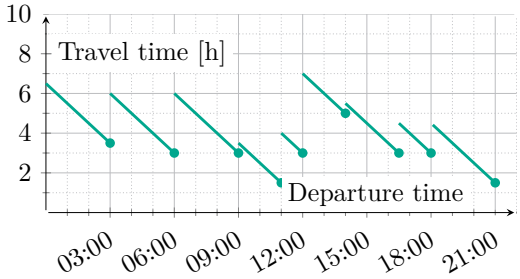

Figure 11: Travel time function on an arc.

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* [177]. 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 [174].

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 [208], in short TE (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.

On time-dependent graphs, Dijkstra's algorithm can be augmented to compute shortest paths [59, 94], as long as the cost functions are nonnegative and FIFO [182, 183]. 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 TD (time-dependent Dijkstra).

Another approach is to exploit the fact that the time-expanded graph is directed and acyclic. 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 [165], 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) [89] 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.

Range Problem. The range problem can be solved on the time-dependent model by variants of Dijkstra's algorithm. The first variant [62, 180] 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 *links* the full travel-time function associated with u to the (time-dependent) cost function of the arc (u, v). The resulting function is then *merged* into the (tentative) travel time function associated with v. 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 [31] 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) [72] 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.

Finally, the Connection Scan Algorithm has been extended to the range problem [89]. 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. The multicriteria problem has received quite some attention in the literature. Computing Pareto sets of shortest paths in (general) graphs can be done by extensions of Dijkstra's algorithm; see [98] for a survey on multicriteria combinatorial optimization. More specifically, the *Multicriteria Label-Setting* (MLS) algorithm [131, 161, 170, 216] 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 [62, 84] 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.

In the time-expanded model, MLS has been considered in a framework called PARETO by Müller-Hannemann and Schnee [174], optimizing arrival time, ticket cost, and number of transfers. In the time-dependent model, Pyrga et al. [193] compute Pareto sets of journeys for arrival time and number of transfers. Disser et al. [92] propose three optimizations to MLS that reduce the number of queue operations: hopping reduction, label forwarding, and dominance by early results (or *target pruning*).

Hansen [131] observes that Pareto sets may contain exponentially many solutions, even for the restricted case of two optimization criteria. In practice, Pareto sets observed in public route planning are much smaller, but can still be large enough to be a bottleneck in computations [177, 178]. To accelerate the query, one can compute approximate solutions, for example, by relaxing domination. In particular, $(1 + \varepsilon)$-Pareto sets have provable polynomial size [186] and can be computed efficiently [156, 219, 225]. For the case of optimizing earliest arrival time and number of transfers, the *Layered Dijkstra* (LD) algorithm [51, 193] is more efficient. Given an upper bound K on the number of transfers, it (implicitly) copies the 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.

A different approach is *RAPTOR* (Round-bAsed Public Transit Optimized Router) [78]. It is explicitly developed for public transit networks and optimizes—in its basic version—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. It 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).

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 Figure 8). Unfortunately, the speedups observed in public transit networks are several orders of magnitude lower than on road networks. This is to some extent explained by the quite different structural properties of public transit and road networks [20]. 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 Section 4.1). Instead, such studies simply perform point-to-point queries on public-transit graphs. In particular, Holzer et al. [137] evaluate basic combinations of bidirectional search, goal directed search, and Geometric Containers on a simple stop graph (with average travel times). Bauer et al. [38] 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 [37].

A* Search. On public transit networks, basic A* search has been applied to the time-dependent model [92,193]. In the context of multicriteria optimization, Disser et al. [92] 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 [125] algorithm has been adapted to both the time-expanded [76] and the time-dependent [181] 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 [208, 224] 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 Section 2, bounding boxes perform best [224].

Arc Flags and SHARC. Delling et al. [76] have adapted Arc Flags [133, 152] 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 [76], making the choice of tie-breaking rules important. Furthermore, Delling et al. [76] 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 [36], has been tested on the time-dependent model with earliest arrival queries by Delling [63]. 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 [43]. 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 [63]. 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" [43]. 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. Then some additional pruning is possible during search [45].

Overlay Graphs. To accelerate public transit queries, Schulz et al. [208] compute single-level overlays between "important" hub stations in the time-expanded model, with importance values given as input. Extending this approach to multiple levels of hub stations (selected by importance or degree) results in speedups of about 11 [209]. A systematic experimental study of overlay-based methods, including time-expanded transit networks, was conducted by Holzer et al. [136].

Separator-based techniques. Strasser and Wagner [215] combine the Connection Scan Algorithm [89] with basic ideas of customizable route planning (CRP) [67]. The Accelerated Connection Scan Algorithm (ACSA) is designed for both earliest arrival and profile queries. As shown in Section 4.5, it achieves excellent query and preprocessing times on country-sized instances.

Contraction Hierarchies. The Contraction Hierarchies algorithm [119] has been adapted to the realistic time-dependent model with minimum change times for computing earliest arrival and range queries [113]. 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* [22]. 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 precomputed for all pairs of stops and departure times. At query time, a query graph is built from 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.

Precomputing transfer patterns between *all* pairs of stops is prohibitive in practice. Therefore, a two-level approach 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. Even with this optimization, preprocessing is impractical on continent-sized networks. It becomes feasible when restricting the local transfer patterns to at most three legs (two transfers). Although this restriction is heuristic, the algorithm almost always

finds the optimal solution in practice, since journeys requiring more than two transfers to reach a hub station are rare.

TRANSIT. Finally, Transit Node Routing [25, 27, 197] has been adapted to public transit journey planning in [12]. Preprocessing of the resulting *TRANSIT* algorithm uses the (small) stop graph to determine a set of transit nodes (with a similar method as in [25]), 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 Section 4.1, extended scenarios (such as incorporating delays) have been studied as well.

Uncertainty and Delays. Trains, busses 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. [105] 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. Müller-Hannemann and Schnee [175] 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. Berger et al. [44] propose a realistic stochastic model that predicts how such delays propagate through the network. In particular, this model is evaluated using real (delay) data from Deutsche Bahn. Bast et al. [23] 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. [92] and Delling et al. [79] study the computation of *reliable* journeys via multi-criteria 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. [89] 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. [123] 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 [122]. A different, new robustness concept has been proposed by Böhmová et al. [47]. 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. [129] have considered the computation of overnight train journeys, whose optimization goals are quite different from regular "daytime" 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 [173] 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 [173].

Delling et al. [78] 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 [24] 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 profile multicriteria 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 exception is 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 number of elementary connections (as a proxy for its size). 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 format[2]. Still, research groups often interpret the data differently, making it hard to compare the performance of different algorithms. The largest metropolitan instance is currently available the full transit network of London[3]. 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 Madrid, Germany and long-distance trains in Europe are generated in a similar way, but the North America instance incorporates further realistic modeling aspects such as extensive footpaths or a two-week schedule (instead of 24 hours).

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. References indicate the publications from which the figures are taken (which may differ from the first publication); TE was run by the authors for this survey. 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 24-hour range queries between 0:00 and 24:00 (rng). Methods with multiple criteria compute Pareto sets.

Among algorithms without preprocessing, observe that those that do not use a graph (RAPTOR and CSA) are consistently faster than their graph-based counterparts. Moreover, using the time-expanded graph model (TE) is significantly slower than the time-dependent graph model (TD), 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 Transit Patterns) can 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.

For multicriteria queries, Transfer Patterns and RAPTOR are the fastest algorithms for metropolitan-sized networks. RAPTOR is twice as fast as TD, 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. For continental networks, Transfer Patterns is the fastest algorithm. Since North America is a very large instance and uses a two-week schedule, Transfer Patterns uses the three-leg heuristic combined with hub stops to reduce the preprocessing effort (space and time). It is still quite accurate in practice, finding optimal results for almost all queries. The error rate is only 0.03% for Switzerland [22] and 0.01% for New York [21], and the few suboptimal journeys are very close to optimal. For the smaller Madrid instance, quadratic (in the number of stops) preprocessing effort is feasible, so these optimizations are not necessary and the algorithm is guaranteed to find

[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, and its number of elementary connections (in millions). It then shows the criteria that are optimized (a subset of arrival times, transfers, 24-hour 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). An asterisk (*) indicates runs on extended inputs with slightly suboptimal results on a very small fraction of all queries; see main text for details.

		INPUT		CRITERIA						QUERY		
algorithm	*impl.*	name	conn. [10^6]	*arr.*	*tran.*	*rng.*	*fare*	*rel.*	prep. [h]	comp. /stop	jn.	time [ms]
TE		London	5.1	•	○	○	○	○	–	50.6	0.9	44.8
TD	[79]	London	5.1	•	○	○	○	○	–	7.4	0.9	11.0
CH	[113]	Europe (LD)	1.7	•	○	○	○	○	<0.1	<0.1	n/a	0.3
CSA	[89]	London	4.9	•	○	○	○	○	–	26.6	n/a	2.0
ACSA	[215]	Germany	46.2	•	○	○	○	○	–	n/a	n/a	8.7
T. Patterns	[24]	Madrid	4.8	•	○	○	○	○	19	–	n/a	0.7
LD	[79]	London	5.1	•	•	○	○	○	–	15.6	1.8	28.7
MLS	[79]	London	5.1	•	•	○	○	○	–	23.7	1.8	50.0
RAPTOR	[79]	London	5.1	•	•	○	○	○	–	10.9	1.8	5.4
T. Patterns	[24]	Madrid	4.8	•	•	○	○	○	185	–	n/a	3.1
T. Patterns*	[22]	N. America	56.6	•	•	○	○	○	2304	–	n/a	12.0
CH	[113]	Europe (LD)	1.7	•	○	•	○	○	<0.1	<0.1	n/a	27.0
SPCS	[89]	London	4.9	•	○	•	○	○	–	372.5	98.2	843.0
CSA	[89]	London	4.9	•	○	•	○	○	–	436.9	98.2	161.0
ACSA	[215]	Germany	46.2	•	○	•	○	○	8	n/a	n/a	171.0
rRAPTOR	[89]	London	4.9	•	•	•	○	○	–	1634.0	203.4	922.0
CSA	[89]	London	4.9	•	•	•	○	○	–	3824.9	203.4	466.0
T. Patterns	[24]	Madrid	4.8	•	•	•	○	○	185	–	n/a	3.1
MLS	[79]	London	5.1	•	•	○	•	○	–	818.2	8.8	304.2
McRAPTOR	[79]	London	5.1	•	•	○	•	○	–	277.5	8.8	100.9
MLS	[79]	London	5.1	•	•	○	○	•	–	286.6	4.7	239.8
McRAPTOR	[79]	London	5.1	•	•	○	○	•	–	89.6	4.7	71.9

the optimal solution. Although preprocessing can be quite heavy for Transfer Patterns, the data it produces is robust even against large and area-wide delays, resulting in much less than 1 % of suboptimal journeys [23].

For range queries, preprocessing-based techniques (CH and ACSA) scale better than CSA or SPCS. For full 24-hour 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 [89], the performance of rRAPTOR depends linearly on the number of journeys departing within the time range [78]. For example, for 2-hour range queries rRAPTOR computes 15.9 journeys taking only 61.3 ms on average [79] (not reported in the table). Guidebook routes covering about 80 % of the optimal results (for the full 24-hour period) can be computed in a fraction of a millisecond [24].

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 Section 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 other 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 [75, 184, 229]. Typical examples [75, 184] model car travel and walking as time-independent (static) graphs, public transit networks using the realistic time-dependent model [193], and flight networks using a dedicated flight model [77]. Beyond that, Kirchler et al. [144, 145] 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 Section 2.7 for details).

Overview. The remainder of this section discusses three different approaches to the multimodal problem. The first (Section 5.1) considers a combined cost function of travel time with some penalties to account for modal transfers. The second approach (Section 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 (Section 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. [9], who present a linear program that computes multimodal journeys. The TRANSIT algorithm [12] also uses a linear utility function and incorporates travel time, ticket cost, and "inconvenience" of transfers. Finally, Modesti and Sciomachen [169] 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* [19] 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 [19], which suffice to model reasonable transport mode constraints in multimodal journey planning [16, 18]. Even restricted classes of regular languages can be useful, such as those that impose a hierarchy of transport modes [46, 75, 144, 145, 184, 229] or Kleene languages that can only globally exclude (and include) certain transport modes [117].

Barrett et al. [19] 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* (LCSPP-D), first builds a product network $\boldsymbol{G}$ of the input (the multimodal graph) and the (possibly nondeterministic) 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 $\boldsymbol{G}$, containing those product vertices that refer to s/t and an initial/final state of the automaton. Dijkstra's algorithm is then run on $\boldsymbol{G}$ between these two sets of product vertices. In a follow-up experimental study, Barrett et al. [18] evaluate this algorithm using linear regular languages, a special case.

Basic speedup techniques, such as bidirectional search [61], A* [132], and heuristic A* [210] have been evaluated in the context of multimodal journey planning in [135] and [17] Also, Pajor [184] combines the LCSPP-D algorithm with time-dependent Dijkstra [59] to compute multimodal journeys that contain a time-dependent subnetwork. He also adapts and analyzes bidirectional search [61], ALT [125], Arc Flags [133, 152], and shortcuts [222] with respect to LCSPP.

Access-Node Routing. The *Access-Node Routing* (ANR) [75] algorithm is a speedup technique for the label-constrained shortest path problem (LCSPP). 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 [25–27, 197] 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* [75] method further reduces preprocessing space and time by combining ANR with contraction. As in Core-ALT [37, 74], 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 [75]. 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 LCSPP is *State-Dependent ALT* (SDALT) [145]. It augments the ALT algorithm [125] 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 [144]. SDALT has been evaluated [144, 145] 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. [90] have adapted Contraction Hierarchies [119] to LCSPP, 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 LCSPP-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 [21, 64]. 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. [64] 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 [131, 161, 170, 216] 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 [90]. To get better query performance, they extend RAPTOR [78] to the multimodal scenario, which results in the *multimodal multicriteria RAPTOR* algorithm (MCR) [64]. 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 [230]. 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. [21] 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 minutes), the rules can then be applied to filter the reasonable journeys. As in [64], 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 [233] 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 [66].

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 [204]. Microsoft's Bing Maps[4] use CRP for routing on road networks. OSRM [159], a popular route planning engine using OpenStreetMap data, uses CH for queries. The Transfer Patterns [22] algorithm has been in use for public-transit journey planning on Google Maps[5] since 2010. RAPTOR is currently in use by OpenTripPlanner[6].

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 Rome2Rio[7] 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.

References

[1] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. VC-dimension and shortest path algorithms. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP'11)*, volume 6755 of *Lecture Notes in Computer Science*, pages 690–699. Springer, 2011.

[2] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. HLDB: Location-based services in databases. In *Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS'12)*, pages 339–348. ACM Press, 2012.

[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`

[3] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension and provably efficient shortest path algorithms. Technical Report MSR-TR-2013-91, Microsoft Research, 2013.

[4] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A hub-based labeling algorithm for shortest paths on road networks. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011.

[5] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical hub labelings for shortest paths. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

[6] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Alternative routes in road networks. *ACM Journal of Experimental Algorithmics*, 18(1):1–17, 2013.

[7] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 782–793. SIAM, 2010.

[8] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–223, 1990.

[9] Georgia Aifadopoulou, Athanasios Ziliaskopoulos, and Evangelia Chrisohoou. Multiobjective optimum path algorithm for passenger pretrip planning in multimodal transportation networks. *Journal of the Transportation Research Board*, 2032(1):26–34, December 2007. 10.3141/2032-04.

[10] Takuya Akiba, Yoichi Iwata, Ken ichi Kawarabayashi, and Yuki Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 147–154. SIAM, 2014.

[11] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

[12] Leonid Antsfeld and Toby Walsh. Finding multi-criteria optimal paths in multi-modal public transportation networks using the transit algorithm. In *Proceedings of the 19th ITS World Congress*, 2012.

[13] Julian Arz, Dennis Luxen, and Peter Sanders. Transit node routing reconsidered. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013.

[14] Maxim Babenko, Andrew V. Goldberg, Anupam Gupta, and Viswanath Nagarajan. Algorithms for hub label optimization. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 69–80. Springer, 2013.

[15] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS'11)*, volume 6595 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2011.

[16] Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav V. Marathe, and Dorothea Wagner. Engineering label-constrained shortest-path algorithms. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 27–37. Springer, June 2008.

[17] Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav V. Marathe, and Dorothea Wagner. Engineering label-constrained shortest-path algorithms. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 309–319. American Mathematical Society, 2009.

[18] Chris Barrett, Keith Bisset, Riko Jacob, Goran Konjevod, and Madhav V. Marathe. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02)*, volume 2461 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2002.

[19] Chris Barrett, Riko Jacob, and Madhav V. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

[20] Hannah Bast. Car or public transport – two worlds. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2009.

[21] Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result diversity for multi-modal route planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 123–136, September 2013.

[22] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

[23] Hannah Bast, Jonas Sternisko, and Sabine Storandt. 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'13)*, OpenAccess Series in Informatics (OASIcs), pages 42–54, September 2013.

[24] Hannah Bast and Sabine Storandt. Flow-based guidebook routing. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 155–165. SIAM, 2014.

[25] Holger Bast, Stefan Funke, and Domagoj Matijevic. Ultrafast shortest-path queries via transit nodes. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 175–192. American Mathematical Society, 2009.

[26] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant shortest-path queries in road networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.

[27] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

[28] Gernot Veit Batz, Robert Geisberger, Dennis Luxen, Peter Sanders, and Roman Zubkov. Efficient route compression for hybrid route planning. In *Proceedings of the 1st Mediterranean Conference on Algorithms*. Springer, 2012.

[29] Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, April 2013.

[30] Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*. Springer, 2012.

[31] Andreas Bauer. Multimodal profile queries. Bachelor thesis, Karlsruhe Institute of Technology, May 2012.

[32] Reinhard Bauer, Moritz Baum, Ignaz Rutter, and Dorothea Wagner. On the complexity of partitioning graphs for arc-flags. *Journal of Graph Algorithms and Applications*, 17(3):265–299, 2013.

[33] Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing speed-up techniques is hard. In *Proceedings of the 7th Conference on Algorithms and Complexity (CIAC'10)*, volume 6078 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2010.

[34] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2013.

[35] Reinhard Bauer, Gianlorenzo D'Angelo, Daniel Delling, Andrea Schumm, and Dorothea Wagner. The shortcut problem – complexity and algorithms. *Journal of Graph Algorithms and Applications*, 16(2):447–481, 2012.

[36] Reinhard Bauer and Daniel Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008.

[37] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, January 2010. Special Section devoted to WEA'08.

[38] Reinhard Bauer, Daniel Delling, and Dorothea Wagner. Experimental study on speed-up techniques for timetable information systems. *Networks*, 57(1):38–52, January 2011.

[39] Reinhard Bauer, Marcus Krug, Sascha Meinert, and Dorothea Wagner. Synthetic road networks. In *Proceedings of the 6th International Conference on Algorithmic Aspects in Information and Management (AAIM'10)*, volume 6124 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2010.

[40] Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

[41] Norbert Baumann and Richard Schmidt. Buxtehude–Garmisch in 6 Sekunden. Die elektronische Fahrplanauskunft (EFA) der Deutschen Bundesbahn. *Zeitschrift für aktuelle Verkehrsfragen*, 10:929–931, 1988.

[42] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[43] Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. 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'09)*, OpenAccess Series in Informatics (OASIcs), 2009.

[44] Annabell Berger, Andreas Gebhardt, Matthias Müller–Hannemann, and Martin Ostrowski. Stochastic delay prediction in large train networks. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–111, 2011.

[45] Annabell Berger, Martin Grimmer, and Matthias Müller–Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.

[46] Maurizio Bielli, Azedine Boulmakoul, and Hicham Mouncif. Object modeling and path computation for multimodal travel systems. *European Journal of Operational Research*, 175(3):1705–1730, 2006.

[47] Katerina Bohmova, Matúš Mihal'ák, Tobias Pröger, Rastislav Šrámek, and Peter Widmayer. 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'13)*, OpenAccess Series in Informatics (OASIcs), pages 27–41, September 2013.

[48] Adi Botea. Ultra-fast optimal pathfinding without runtime search. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'11)*, pages 122–127. AAAI Press, 2011.

[49] Adi Botea and Daniel Harabor. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*. AAAI Press, 2013.

[50] Ulrik Brandes, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Travel planning with self-made maps. In *Proceedings of the 3rd International Workshop on Algorithm Engineering and Experiments (ALENEX'01)*, volume 2153 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2001.

[51] Gerth Brodal and Riko Jacob. 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'03)*, volume 92 of *Electronic Notes in Theoretical Computer Science*, pages 3–15, 2004.

[52] Francesco Bruera, Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, and Daniele Frigioni. Dynamic multi-level overlay graphs for shortest paths. *Mathematics in Computer Science*, 1(4):709–736, April 2008.

[53] Edith Brunel, Daniel Delling, Andreas Gemsa, and Dorothea Wagner. Space-efficient sharc-routing. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 47–58. Springer, May 2010.

[54] Tom Caldwell. On finding minimum routes in a network with turn penalties. *Communications of the ACM*, 4(2):107–108, 1961.

[55] Cambridge Vehicle Information Technology Ltd. Choice Routing, 2005. Available at `http://www.camvit.com`.

[56] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms. *Mathematical Programming, Series A*, 73:129–174, 1996.

[57] Boris V. Cherkassky, Andrew V. Goldberg, and Craig Silverstein. Buckets, heaps, lists, and monotone priority queues. In *Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 83–92. IEEE Computer Society Press, 1997.

[58] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[59] K. Cooke and E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

[60] Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, and Camillo Vitale. Fully dynamic maintenance of arc-flags in road networks. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2012.

[61] George B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1962.

[62] Brian C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[63] Daniel Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, May 2011.

[64] Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing multimodal journeys in practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2013.

[65] Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck. PHAST: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, 73(7):940–952, 2013.

[66] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.

[67] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. submitted for publication, 2013.

[68] Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. Graph partitioning with natural cuts. In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 1135–1146. IEEE Computer Society, 2011.

[69] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Faster batched shortest paths in road networks. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 52–63, 2011.

[70] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hub label compression. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2013.

[71] Daniel Delling, Martin Holzer, Kirill Müller, Frank Schulz, and Dorothea Wagner. High-performance multi-level routing. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 73–92. American Mathematical Society, 2009.

[72] Daniel Delling, Bastian Katz, and Thomas Pajor. Parallel computation of best connections in public transportation networks. *ACM Journal of Experimental Algorithmics*, 17(4):4.1–4.26, July 2012.

[73] Daniel Delling, Moritz Kobitzsch, Dennis Luxen, and Renato F. Werneck. Robust mobile route planning with limited connectivity. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 150–159. SIAM, 2012.

[74] Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road networks. *Informs Journal on Computing*, 24(2):187–201, 2012.

[75] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multi-modal route planning by access-nodes. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 587–598. Springer, September 2009.

[76] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 182–206. Springer, 2009.

[77] Daniel Delling, Thomas Pajor, Dorothea Wagner, and Christos Zaroliagis. Efficient route planning in flight networks. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, OpenAccess Series in Informatics (OASIcs), 2009.

[78] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

[79] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 2014. submitted.

[80] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

[81] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Highway hierarchies star. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 141–174. American Mathematical Society, 2009.

[82] Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *Lecture Notes in Computer Science*, pages 52–65. Springer, June 2007.

[83] Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, June 2009.

[84] Daniel Delling and Dorothea Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

[85] Daniel Delling and Renato F. Werneck. Customizable point-of-interest queries in road networks. In *Proceedings of the 21st ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS'13)*. ACM Press, 2013. to appear.

[86] Daniel Delling and Renato F. Werneck. Faster customization of road networks. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2013.

[87] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society, 2009.

[88] Eric V. Denardo and Bennett L. Fox. Shortest-route methods: 1. Reaching, pruning, and buckets. *Operations Research*, 27(1):161–186, 1979.

[89] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

[90] Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-constrained multi-modal route planning. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129. SIAM, 2012.

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

[92] Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

[93] Florian Drews and Dennis Luxen. Multi-hop ride sharing. In *Proceedings of the 5th International Symposium on Combinatorial Search (SoCS'12)*, pages 71–79. AAAI Press, 2013.

[94] Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

[95] Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 25:25–25:30. ACM Press, November 2013.

[96] Alexandros Efentakis, Dieter Pfoser, and Agnes Voisard. Efficient data management in support of shortest-path computation. In *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 28–33. ACM Press, 2011.

[97] Alexandros Efentakis, Dimitris Theodorakis, and Dieter Pfoser. Crowdsourcing computing resources for shortest-path computation. In *Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS'12)*, pages 434–437. ACM Press, 2012.

[98] Matthias Ehrgott and Xavier Gandibleux, editors. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. Kluwer Academic Publishers Group, 2002.

[99] David Eisenstat. Random road networks: The quadtree model. In *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics (ANALCO '11)*, pages 76–84. SIAM, January 2011.

[100] Jochen Eisner and Stefan Funke. 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'12)*, pages 502–505. ACM Press, 2012.

[101] Jochen Eisner and Stefan Funke. Transit nodes – Lower bounds and refined construction. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 141–149. SIAM, 2012.

[102] Jochen Eisner, Stefan Funke, Andre Herbst, Andreas Spillner, and Sabine Storandt. Algorithms for matching and predicting trajectories. In *Proceedings of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX'11)*, pages 84–95. SIAM, 2011.

[103] Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large network. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, August 2011.

[104] David Eppstein and Michael T. Goodrich. 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 '08)*, pages 1–10. ACM Press, 2008.

[105] Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is timetabling routing always reliable for public transport? In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 15–26, September 2013.

[106] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[107] Lester R. Ford, Jr. Network flow theory. Technical Report P-923, Rand Corporation, Santa Monica, California, 1956.

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

[109] L. Fu, D. Sun, and L. R. Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33(11):3324–3343, 2006.

[110] Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments (ALENEX'13)*, pages 31–54. SIAM, 2013.

[111] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2–3):111–120, September 2003.

[112] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53:85–112, 2004.

[113] Robert Geisberger. Contraction of timetable networks with realistic transfers. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, May 2010.

[114] Robert Geisberger. *Advanced Route Planning in Transportation Networks*. PhD thesis, Karlsruhe Institute of Technology, February 2011.

[115] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 124–137. SIAM, 2010.

[116] Robert Geisberger, Dennis Luxen, Peter Sanders, Sabine Neubauer, and Lars Volker. Fast detour computation for ride sharing. In *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'10)*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 88–99, 2010.

[117] Robert Geisberger, Michael Rice, Peter Sanders, and Vassilis Tsotras. Route planning with flexible edge restrictions. *ACM Journal of Experimental Algorithmics*, 17(1):1–20, 2012.

[118] Robert Geisberger and Peter Sanders. 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'10)*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, 2010.

[119] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[120] Robert Geisberger and Dennis Schieferdecker. Heuristic contraction hierarchies with approximation guarantee. In *Proceedings of the 3rd International Symposium on Combinatorial Search (SoCS'10)*. AAAI Press, 2010.

[121] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2011.

[122] Marc Goerigk, Sascha Heße, Matthias Müller–Hannemann, and Marie Schmidt. Recoverable robust timetable information. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 1–14, September 2013.

[123] Marc Goerigk, Martin Knoth, Matthias Müller–Hannemann, Marie Schmidt, and Anita Schöbel. The price of strict and light robustness in timetable information. *Transportation Science*, 2013. Published online before print.

[124] Andrew V. Goldberg. A practical shortest path algorithm with linear expected time. *SIAM Journal on Computing*, 37:1637–1655, 2008.

[125] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

[126] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for A*: Shortest path algorithms with preprocessing. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 93–139. American Mathematical Society, 2009.

[127] Andrew V. Goldberg and Renato F. Werneck. Computing point-to-point shortest paths from external memory. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 26–40. SIAM, 2005.

[128] Roy Goldman, N.R. Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB 1998)*, pages 26–37. Morgan Kaufmann, August 1998.

[129] Thorsten Gunkel, Mathias Schnee, and Matthias Müller–Hannemann. How to find good night train connections. *Networks*, 57(1):19–27, 2011.

[130] Ronald J. Gutman. 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'04)*, pages 100–111. SIAM, 2004.

[131] Pierre Hansen. Bricriteria path problems. In *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.

[132] Peter E. Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[133] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 41–72. American Mathematical Society, 2009.

[134] Petr Hliněný and Ondrej Moriš. Scope-based route planning. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA'11)*, volume 6942 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2011.

[135] Martin Holzer. *Engineering Planar-Separator and Shortest-Path Algorithms*. PhD thesis, Karlsruhe Institute of Technology (KIT) - Department of Informatics, 2008.

[136] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

[137] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *ACM Journal of Experimental Algorithmics*, 10(2.5):1–18, 2006.

[138] Eric Horvitz and John Krumm. Some help on the way: Opportunistic routing under uncertainty. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (Ubicomp'12)*, pages 371–380. ACM Press, 2012.

[139] T. Ikeda, Min-Yao Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by AI search techniques. In *Proceedings of the Vehicle Navigation and Information Systems Conference (VNSI'94)*, pages 291–296. ACM Press, 1994.

[140] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):409–432, May 1998.

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

[142] Hermann Kaindl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7:283–317, December 1997.

[143] Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed time-dependent contraction hierarchies. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 83–93. Springer, May 2010.

[144] Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. A label correcting algorithm for the shortest path problem on a multi-modal route network. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2012.

[145] Dominik Kirchler, Leo Liberti, Thomas Pajor, and Roberto Wolfler Calvo. 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'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 64–75, 2011.

[146] Jon M. Kleinberg, Aleksandrs Slivkins, and Tom Wexler. Triangulation and embedding using small sets of beacons. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 444–453. IEEE Computer Society Press, 2004.

[147] Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing many-to-many shortest paths using highway hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

[148] Moritz Kobitzsch. Hidar: An alternative approach to alternative routes. In *Proceedings of the 21st Annual European Symposium on Algorithms (ESA'13)*, volume 8125 of *Lecture Notes in Computer Science*, pages 613–624. Springer, 2013.

[149] Moritz Kobitzsch, Marcel Radermacher, and Dennis Schieferdecker. 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'13)*, OpenAccess Series in Informatics (OASIcs), pages 94–107, September 2013.

[150] John Krumm, Robert Gruen, and Daniel Delling. From destination prediction to route prediction. *Journal of Location Based Services*, 7(2):98–120, 2013.

[151] John Krumm and Eric Horvitz. Predestination: Where do you want to go today? *IEEE Computer*, 40(4):105–107, 2007.

[152] Ulrich Lauther. 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*, volume 74 of *DIMACS Book*, pages 19–40. American Mathematical Society, 2009.

[153] Ken C.K. Lee, Jian Lee, Baihua Zheng, and Yuan Tian. ROAD: A new spatial object search framework for road networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(3):547–560, November 2012.

[154] Richard J. Lipton, Donald J. Rose, and Robert Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, April 1979.

[155] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, April 1979.

[156] P Loridan. ϵ-solutions in vector minimization problems. *Journal of Optimization Theory and Applications*, 43(2):265–276, 1984.

[157] Dennis Luxen and Peter Sanders. Hierarchy decomposition for faster user equilibria on road networks. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 242–253. Springer, 2011.

[158] Dennis Luxen and Dennis Schieferdecker. Candidate sets for alternative routes in road networks. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 260–270. Springer, 2012.

[159] Dennis Luxen and Christian Vetter. Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Press, 2011.

[160] Kamesh Madduri, David A. Bader, Jonathan W. Berry, and Joseph R. Crobak. Parallel shortest path algorithms for solving large-scale instances. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 249–290. American Mathematical Society, 2009.

[161] Ernesto Queiros Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

[162] Jens Maue, Peter Sanders, and Domagoj Matijevic. Goal-directed shortest-path queries using precomputed cluster distances. *ACM Journal of Experimental Algorithmics*, 14:3.2:1–3.2:27, 2009.

[163] Kurt Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[164] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[165] T. Mellouli and L. Suhl. Passenger online routing in dynamic networks. In D. C. Mattfeld and L. Suhl, editors, *Informationsprobleme in Transport und Verkehr*, volume 4, pages 17–30. DS&OR Lab, Universität Paderborn, 2006.

[166] Ulrich Meyer. 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'01)*, pages 797–806, 2001.

[167] Ulrich Meyer and Peter Sanders. δ-stepping: A parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[168] Nikola Milosavljević. On optimal preprocessing for contraction hierarchies. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 33–38. ACM Press, 2012.

[169] Paola Modesti and Anna Sciomachen. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *European Journal of Operational Research*, 111(3):495–508, 1998.

[170] Rolf H. Möhring. Verteilte Verbindungssuche im öffentlichen Personenverkehr – Graphentheoretische Modelle und Algorithmen. In *Angewandte Mathematik insbesondere Informatik, Beispiele erfolgreicher Wege zwischen Mathematik und Informatik*, pages 192–220. Vieweg, 1999.

[171] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup Dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, 11(2.8):1–29, 2006.

[172] Edward F. Moore. The Shortest Path Through a Maze. In *Proc. of the Int. Symp. on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[173] Matthias Müller–Hannemann and Mathias Schnee. Paying less for train connections with motis. In *Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, OpenAccess Series in Informatics (OASIcs), page 657, 2006.

[174] Matthias Müller–Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[175] Matthias Müller–Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 249–272. Springer, 2009.

[176] Matthias Müller–Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2007.

[177] Matthias Müller–Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

[178] Matthias Müller–Hannemann and Karsten Weihe. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

[179] Laurent Flindt Muller and Martin Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'07)*, volume 4698 of *Lecture Notes in Computer Science*, pages 657–668. Springer, 2007.

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

[181] Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, 59:240–251, 2012.

[182] Ariel Orda and Raphael Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[183] Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21:295–319, 1991.

[184] Thomas Pajor. Multi-modal route planning. Master's thesis, Universität Karlsruhe (TH), March 2009.

[185] Stefano Pallottino and Maria Grazia Scutellà. Shortest path algorithms in transportation models: Classical and innovative aspects. In *Equilibrium and Advanced Transportation Modelling*, pages 245–281. Kluwer Academic Publishers Group, 1998.

[186] Christos H. Papadimitriou and Mihalis Yannakakis. 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'00)*, pages 86–92, 2000.

[187] Andreas Paraskevopoulos and Christos Zaroliagis. Improved alternative route planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 108–122, September 2013.

[188] Seymour V. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, April 1961.

[189] David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000.

[190] Ira Pohl. Bi-directional and heuristic search in path problems. Technical Report SLAC-104, Stanford Linear Accelerator Center, Stanford, California, 1969.

[191] Ira Pohl. Bi-directional search. In *Proceedings of the Sixth Annual Machine Intelligence Workshop*, volume 6, pages 124–140. Edinburgh University Press, 1971.

[192] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Experimental comparison of shortest path approaches for timetable information. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 88–99. SIAM, 2004.

[193] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

[194] Michael Rice and Vassilis Tsotras. Bidirectional A* search with additive approximation bounds. In *Proceedings of the 5th International Symposium on Combinatorial Search (SoCS'12)*. AAAI Press, 2012.

[195] Michael Rice and Vassilis Tsotras. Exact graph search algorithms for generalized traveling salesman path problems. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2012.

[196] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

[197] Peter Sanders and Dominik Schultes. Robust, almost constant time shortest-path queries in road networks. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 193–218. American Mathematical Society, 2009.

[198] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. *ACM Journal of Experimental Algorithmics*, 17(1):1–40, 2012.

[199] Peter Sanders, Dominik Schultes, and Christian Vetter. Mobile route planning. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 732–743. Springer, September 2008.

[200] Peter Sanders and Christian Schulz. Distributed evolutionary graph partitioning. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 16–29. SIAM, 2012.

[201] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS'05)*, pages 200–209, 2005.

[202] Jagan Sankaranarayanan and Hanan Samet. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(8), 2010.

[203] Jagan Sankaranarayanan and Hanan Samet. Roads belong in databases. *IEEE Data Engineering Bulletin*, 33(2):4–11, 2010.

[204] Heiko Schilling. TomTom navigation – How mathematics help getting through traffic faster, 2012. Talk given at ISMP.

[205] Robert Schreiber. A new implementation of sparse Gaussian elimination. *ACM Transactions on Mathematical Software*, 8(3):256–276, September 1982.

[206] Dominik Schultes. *Route Planning in Road Networks.* PhD thesis, Universität Karlsruhe (TH), February 2008.

[207] Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79. Springer, June 2007.

[208] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

[209] Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.

[210] Robert Sedgewick and Jeffrey S. Vitter. Shortest paths in Euclidean graphs. *Algorithmica*, 1(1):31–48, 1986.

[211] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4), 2014.

[212] Sabine Storandt. 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*, pages 234–242, 2012.

[213] Sabine Storandt and Stefan Funke. Cruising with a battery-powered vehicle and not getting stranded. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence.* AAAI Press, 2012.

[214] Sabine Storandt and Stefan Funke. Enabling e-mobility: Facility location for battery loading stations. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence.* AAAI Press, 2013.

[215] Ben Strasser and Dorothea Wagner. Connection scan accelerated. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

[216] Dirk Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen.* PhD thesis, Universität Paderborn, 1995.

[217] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In *35th ACM Symposium on Theory of Computing*, pages 149–158, New York, NY, USA, 2003. ACM.

[218] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[219] George Tsaggouris and Christos Zaroliagis. Multiobjective optimization: Improved fptas for shortest paths and non-linear objectives with applications. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'06)*, volume 4288 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 2006.

[220] Eduard Tulp and Laurent Siklóssy. TRAINS, An Active Time-Table Searcher. In *ECAI*, volume 88, pages 170–175, 1988.

[221] Eduard Tulp and Laurent Siklóssy. Searching Time-Table Networks. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 5(3):189–198, 1991.

[222] Dirck van Vliet. Improved shortest path algorithms for transport networks. *Transportation Research Part B: Methodological*, 12(1):7–20, 1978.

[223] Dorothea Wagner and Thomas Willhalm. Drawing graphs to speed up shortest-path computations. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 15–24. SIAM, 2005.

[224] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM Journal of Experimental Algorithmics*, 10(1.3):1–30, 2005.

[225] Douglas J White. Epsilon efficiency. *Journal of Optimization Theory and Applications*, 49(2):319–337, 1986.

[226] J.W.J. Williams. Algorithm 232: Heapsort. *Journal of the ACM*, 7(6):347–348, June 1964.

[227] Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

[228] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment*, 5(5):406–417, January 2012.

[229] Haicong Yu and Feng Lu. Advanced multi-modal routing approach for pedestrians. In *2nd International Conference on Consumer Electronics, Communications and Networks*, pages 2349–2352, 2012.

[230] Lotfi A. Zadeh. Fuzzy logic. *IEEE Computer*, 21(4):83–93, 1988.

[231] Tim Zeitz. Weak contraction hierarchies work! Bachelor thesis, Karlsruhe Institute of Technology, 2013.

[232] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for KNN search on road networks. In *Proceedings of the 22nd International Conference on Information and Knowledge Management*, pages 39–48. ACM Press, 2013.

[233] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data (SIGMOD'13)*, pages 857–868. ACM Press, 2013.

[234] Uri Zwick. Exact and approximate distances in graphs – A survey. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA'01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48, 2001.