Shortest-Path Queries in Static Networks

Christian Sommer

csom@csail.mit.edu

We consider the point-to-point (approximate) shortest-path query problem, which is the following generalization of the classical single-source (SSSP) and all-pairs shortest paths (APSP) problems: We are first presented with a network (graph). A so-called preprocessing algorithm may compute certain information (a data structure or index) to prepare for the next phase. After this preprocessing step, applications may ask shortest-path or distance queries, which should be answered as fast as possible.

Due to its many applications in areas such as transportation, networking, and social science, this problem has been considered by researchers from various communities (sometimes under different names): algorithm engineers construct fast route planning methods, database and information systems researchers investigate materialization tradeoffs, query processing on spatial networks, and reachability queries, and theoretical computer scientists analyze distance oracles and sparse spanners. Related problems are considered for compact routing and distance labeling schemes in networking and distributed computing and for metric embeddings in geometry as well.

In this survey, we review selected approaches, algorithms, and results on shortest-path queries from these fields, with the main focus lying on the tradeoff between the index size and the query time. We survey methods for general graphs as well as specialized methods for restricted graph classes, in particular for those classes with arguable practical significance such as planar graphs and complex networks.

Categories and Subject Descriptors: G.2.2 [Discrete Mathematics]: Graph Theory; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory

Additional Key Words and Phrases: shortest path, shortest-path query, distance oracle

1. INTRODUCTION

We review research on algorithms for the point-to-point (approximate) shortest-path query problem, restricted to discrete, static graphs with non-negative edge lengths (also called weights or costs). The only criterion on the optimality of a path shall be its length, which is defined as the sum over all the edges on the path of their lengths. Edge and path lengths can be used to represent various quantities such as travel times, ticket prices, or fuel costs.

The shortest-path problem in general has countless applications; the shortest-path query problem in particular occurs in applications such as route planning and navigation [Zaroliagis 2008; Goldberg et al. 2009; Delling et al. 2009a], Geographic Information Systems (GIS) and intelligent transportation systems [Jing et al. 1996], logistics, traffic simulations [Ziliaskopoulos et al. 1997; Barrett et al. 2002; Raney and Nagel 2004; Baker and Gokhale 2007], computer games [Stout 1999; Bulitko et al. 2010], server selection [Ng and Zhang 2002; Dabek et al. 2004; Costa et al. 2004; Shavitt and Tankel 2008; Eriksson et al. 2009], XML indexing [Schenkel et al. 2004; 2005], proximity search in databases [Goldman et al. 1998], reachability in object databases [Butterworth et al. 1991], packet routing [Schwartz and Stern...
The shortest-path query problem is different from the classical single-source (SSSP) and all-pairs shortest paths (APSP) problems in that there are two stages: preprocessing and answering queries. We are first presented with a network (also termed graph). A so-called preprocessing algorithm may compute certain information (a data structure or index, in the theory community referred to as a distance oracle [Thorup and Zwick 2005]) to prepare for the second phase. After this preprocessing step, applications may ask queries, which should be answered efficiently. In computational geometry, this and similar problems are sometimes called repetitive-mode (as opposed to single-shot) problems [Preparata and Shamos 1985, p. 37].

A lazy solution to the shortest-path query problem is not to precompute any data structure at all but to use an SSSP algorithm [Dijkstra 1959; Fredman and Tarjan 1987] to answer queries. Answering a query then requires time roughly linear in the network size. An eager solution is to precompute the results for all possible queries using an APSP algorithm [Floyd 1962; Warshall 1962; Johnson 1977]. We assume no knowledge about the query distribution. In practice, an application designer may potentially take advantage of different frequencies for user queries, in particular if certain pairs are queried significantly more often than others, or if some pairs are expected to never be queried at all. For example, in a route planning system, one might assume that for most user queries origin and destination are within a few hundred miles (while the maximum distance might be a few thousand miles).

Both solutions have their advantages and disadvantages: for the SSSP strategy, no preprocessing is necessary but the query processing is rather slow; for the APSP strategy, the query execution is extremely fast: one table lookup suffices to obtain the shortest-path distance; but the preprocessing step is expensive and the space consumption is prohibitively large for many real-world networks, spanning millions or even billions of nodes.

In the shortest-path query scenario, we mediate between these two extremes, that is, we analyze the tradeoff between space, preprocessing time, and query time. If the query algorithm is allowed to return an approximate shortest path, the worst-case accuracy (often called stretch) is also an important factor of the tradeoff.

Designing a shortest-path query processing method raises questions such as: How can these data structures be computed efficiently? What amount of storage space is necessary? How much improvement of the query time is possible? How good is the approximation quality of the query result? What are the tradeoffs between pre-computation time, space, query time, and approximation quality?

In this survey, we focus on the tradeoff between space and query time. In the first part, we survey theoretical results on distance oracles for general graphs (Section 2). In the second part, we consider two application scenarios and the corresponding graph classes, namely (i) distance oracles for planar graphs, motivated by route planning problems for road networks (Section 3), and (ii) distance oracles for complex networks, motivated by practical problems in online social networks, web search, computer networking, computational biology, and social science (Section 4).

1.1 Problem Specification

Thorup and Zwick [2005] coined the term distance oracle, which is a data structure that, after preprocessing a graph \( G = (V,E) \), allows for efficient (approximate) distance and shortest-path queries. Let \( \ell \) denote the edge length function \( \ell : E \to \mathbb{R}^+ \), which we assume to be non-negative, i.e., \( \forall e \in E : \ell(e) \geq 0 \).

**Definition 1.1.** An \((\alpha, \beta)-approximate\) distance oracle for a class of graphs \( G \) consists of a data structure and a query algorithm. 

—The **preprocessing time** is the worst-case time required to construct the data structure \( S(G) \) for any \( G \in \mathcal{G} \). For randomized preprocessing algorithms, the preprocessing time is, as usual, defined as the maximum over all \( G \in \mathcal{G} \) of the expected preprocessing time for \( G \).

—The **space complexity** refers to the worst-case size of the data structure for any \( G \in \mathcal{G} \).

After preprocessing \( G = (V,E) \), the data structure \( S \) (which depends on \( G \)) supports (approximate) distance queries for all pairs of nodes \( u,v \in V \), returning a value \( \tilde{d}_S(u,v) \). The query algorithm and its result are characterized as follows.

—The **query time** is the worst-case time required to compute \( \tilde{d}_S(u,v) \) among all \( G \in \mathcal{G} \) and \( u,v \in V \).

—A distance oracle \( S \) is said to have stretch \((\alpha, \beta)\) with \( \alpha \geq 1 \) and \( \beta \geq 0 \) if for all \( G \in \mathcal{G} \) and \( u,v \in V \) its query algorithm satisfies

\[
\ell_G(u,v) \leq \tilde{d}_S(u,v) \leq \alpha \cdot \ell_G(u,v) + \beta,
\]

where \( \ell_G(u,v) \) denotes the shortest-path distance from \( u \) to \( v \) in \( G \). The stretch is also called distortion.

In addition to the worst-case measures, the average or expected query time and stretch may also be of interest. If not explicitly stated otherwise, in this survey, stretch means the worst-case stretch. Note that additive stretch \( \beta > 0 \) is most meaningful for unit-length graphs (\( \forall e \in E : \ell(e) = 1 \)); for more general length functions, we usually have \( \beta = 0 \) and stretch means multiplicative stretch \((\alpha \geq 1)\).

---

1 This survey has been modified from its original version. It has been formatted to fit this journal’s page limit and edited for content. Several references had to be removed for brevity, they can be found in [Sommer 2010].

Let us emphasize that the query time corresponds to the time to compute the shortest-path distance (or an estimate thereof), as opposed to an actual path. For many efficient data structures, the time to actually report a shortest path is dominated by the time required to explicitly output each edge. Obviously, this time must at least be proportional to the number of edges on the path, making comparisons somewhat more difficult. For most methods, after having computed the distance, there is an implicit representation of the path such that each edge can be output efficiently (often in constant time). In the following, query times correspond to the times required to compute (approximate) distances.

An (approximate) distance labeling scheme [Peleg 2000; Gavoille et al. 2004] (inspired by adjacency labels [Kannan et al. 1992]) can be thought of as the distributed version of a distance oracle. The data structure is distributed among the nodes such that each node $u$ is assigned a label $L(u)$. The space complexity is defined as the maximum label length (the average label length is also of interest). At query time, the algorithm is given only the two labels $L(s), L(t)$ of the query nodes $s,t$, respectively, using which it must compute (an estimate of) $d_G(s,t)$. 

1.2 General Techniques

On a high level, many methods use some concept of landmarks, portals, hubs, beacons, seeds, or transit nodes, each corresponding to a set of carefully selected points (often a subset $L \subseteq V$ of the node set), which represent (potentially approximate) shortest paths. In the following, we refer to such nodes as landmarks. We distinguish three typical degrees of freedom:

(1) **Global Landmark Selection**: different methods use different selection strategies to choose a global set of landmarks $L \subseteq V$. Popular strategies include (i) random sampling, (ii) high-degree nodes, and (iii) nodes on separators. Depending on the shortest-path metric (defined by the length/weight/cost function $\ell$), there is also a strategy to (iv) choose those nodes that lie on shortest paths of certain lengths (particularly effective for road networks with edge lengths corresponding to estimates of driving times).

(2) **Local Landmark Selection**: each node $u \in V$ is connected to certain landmarks (potentially to all), which usually means that $u$ stores the shortest-path distance to its landmark set $L(u) \subseteq L$. For some methods there is one distinguished landmark $l_u \in L$ associated with each node $u$. Usually, $l_u$ is chosen as a nearest landmark.

(3) **Distances Among/From/To Landmarks**: methods may differ in how they represent distances among landmarks (sometimes only a subset of all the $L \times L$ distances is represented) and from nodes to landmarks (and vice versa). In general, methods store stars (representing SSSP distances from one node to a subset of nodes) and cliques (representing APSP distances among a subset of nodes), which are of course unions of stars. When we say that oracle constructions mediate between SSSP and APSP, it is not only a “global” analogy, but often also corresponds to “local” decisions on where to use SSSP and where to use APSP.

A potential fourth degree of freedom is to use multiple levels or recursion. For example, many methods use landmarks at various scales.
Furthermore, almost always when a partial execution of an SSSP algorithm is involved, the designers may choose between (a) computing the shortest-path tree at preprocessing time and storing it (as a star), or (b) computing the tree at query time, thereby mediating between space and query time in a rather straightforward way.

2. THEORETICAL RESULTS ON DISTANCE ORACLES FOR GENERAL GRAPHS

For distance oracles applicable to general graphs, the quantitative tradeoff between the space requirements and the approximation quality (stretch) is known up to constant factors. For distance oracles that take advantage of the properties of certain classes of graphs, however, the tradeoff is less well understood: for some classes of sparse graphs such as planar graphs, there are data structures that enable query algorithms to efficiently compute distance estimates of much higher precision than what the tradeoff for general graphs would predict.

In the following, we summarize the known theoretical results for general graphs, with space complexity lower bounds in Section 2.1 and algorithmic upper bounds in Section 2.2.

2.1 Lower Bounds on the Space of Distance Oracles

Known lower bounds on the space requirements of distance oracles are listed in Table I. In the following we consider lower bounds for undirected graphs, which extend to directed graphs. However, for directed graphs, even stronger lower bounds are known: note that any distance oracle for directed graphs with arbitrary finite stretch can also answer reachability queries, for which no worst-case efficient data structure is known [Ajtai and Fagin 1990; Patrascu 2011]. The results for undirected graphs can be summarized as follows.

— The lower bound of Thorup and Zwick [2005, Proposition 5.1] (see also [Matousek 1996] for a similar construction) establishes that, for general graphs, the tradeoffs between space, stretch, and query time of existing distance oracles (Section 2.2) are essentially best possible. More precisely, they prove that any distance oracle with multiplicative stretch \( \alpha < 2k + 1 \) must use space \( \Omega(n^{1+1/k}) \), assuming Erdős’ Girth Conjecture [1964], which is widely believed and partially proven. The girth of a graph is defined as the length of its shortest cycle; the conjecture says that dense graphs with large girth exist (see [Thorup and Zwick 2005, Table II] for an overview of results on the Girth Conjecture).

— For sparse graphs, the situation is less clear. Distance oracles with constant stretch and query time require superlinear space [Sommer et al. 2009]. Higher space lower bounds have been given based on the assumption that constant-time set intersection queries require quadratic space: for multiplicative stretch \( \alpha < 2 \), quadratic space is required [Cohen and Porat 2010; Patrascu and Roditty 2010]; for multiplicative stretch \( \alpha < 3 \), in particular for stretch \( \alpha = 3 - 2/\ell \), space \( \tilde{\Omega}(n^{1+1/(2-1/\ell)}) \) is required [Patrascu et al. 2012].

— For special classes of graphs such as planar, bounded-genus, minor-free, and bounded-doubling-dimension graphs I do not know of non-trivial lower bounds.

\[\text{Asymptotic notation as in } \tilde{O}(\cdot) \text{ or } \tilde{\Omega}(\cdot) \text{ hides polylogarithmic factors in the number of nodes } n.\]

Table I. Space lower bounds of distance oracles for graphs on $n$ nodes and $m$ edges, up to polylogarithmic factors. The table is supposed to be read and interpreted as follows: the lower bound on the space (leftmost column) holds if the conditions on stretch and query time in the second column are met, potentially with further assumptions on the model or conjectures (third column).

<table>
<thead>
<tr>
<th>Lower Bound Space</th>
<th>Condition(s)</th>
<th>Assumption / Proof</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega(\min{m, n^{2/(3+\epsilon)}})$</td>
<td>$\alpha = 2k+1$</td>
<td>girth conjecture [Erdős 1964]</td>
<td>Thorup and Zwick 2005</td>
</tr>
<tr>
<td>$\Omega(n^2)$</td>
<td>$\alpha &lt; 2$</td>
<td>cell-probe model [Yao 1981]</td>
<td>Sommer et al. 2009</td>
</tr>
<tr>
<td>$\tilde{\Omega}(n^{1+\epsilon/(2-\epsilon)})$</td>
<td>$\alpha = 3-2\epsilon$</td>
<td>set intersection, conjecture [Cohen and Porat 2010]</td>
<td>Patrascu et al. 2012</td>
</tr>
<tr>
<td>$\Omega(n^{1/2})$</td>
<td>$\alpha = 1$</td>
<td>distributed labels</td>
<td>Gavoille et al. 2004</td>
</tr>
</tbody>
</table>

**Lower Bounds on the Lengths of Distance Labels.** Since distance labeling schemes are in some sense distributed distance oracles, space lower bounds on oracles extend to bounds on label lengths in a straightforward way. However, lower bounds on label lengths can be higher, since the query algorithm is allowed to access only two labels (as opposed to an entire data structure). Several such lower bounds are due to Gavoille, Peleg, Perennes, and Raz [2004] (see also Section 3.1.1). One of their lower bounds says that, even for graphs with maximum degree 3, exact distance labels require total label length $\Omega(n^{3/2})$ [Gavoille et al. 2004, Theorem 3.7].

**Preprocessing Time.** There is also a connection between boolean matrix multiplication and distance oracles with multiplicative stretch $\alpha < 2$ or additive stretch $\beta = O(1)$. Dor, Halperin, and Zwick [2000, Theorem 5.1] provide a reduction between approximate APSP and matrix multiplication: if the preprocessing algorithm is faster than the time required for matrix multiplication, then query time $o(m/n)$ would imply a faster algorithm for boolean matrix multiplication.

### 2.2 Distance Oracles for General Graphs

For an overview of distance oracles for general undirected graphs, see Table II. For the stretch vs. space tradeoff, see Fig. 1.

Many distance oracles approximate distances by triangulation using a sublinear number of landmarks (also termed beacons), selected by random sampling. Nodes store distances to all landmarks $l \in L$ (stars from all landmarks) and query results are computed as $\min_{l \in L} d(s, l) + d(l, t)$. If any landmark $l \in L$ lies on a shortest path from $s$ to $t$, the exact distance can be recovered. However, most schemes do not guarantee that shortest distances are retrieved. Furthermore, instead of minimizing over all landmarks (in time $O(|L|)$), in many schemes, nodes designate a nearest landmark for triangulation. Let $l_s (l_t)$ denote a landmark that is closest to $s$ (t). Then, the result is simply $\min\{d(s, l_s) + d(l_s, t), d(s, l_t) + d(l_t, t)\}$.

The approximation obtained by triangulation can be rather inaccurate if $s$ and $t$ are close to each other. Triangulation may incur a detour that is arbitrarily large with respect to the shortest-path distance. The approximate distance oracles described in the following differ mainly in their handling of short distances.

**Odd Integral Stretch.** In a seminal work, Thorup and Zwick [2005] provide both the lower and the matching upper bound: for any integer $k \geq 1$ there is a distance
Table II. Time and space complexities of distance oracles for general, undirected graphs, sorted by stretch (more precisely, by the minimum stretch possible). For the oracles in the lower part, the bounds apply for unit-length graphs only and the stretch often also involves an additive term $\beta$. Approximate distance oracles are included only if the space requirement is at most $o(n^2)$.

For $k = 2$ (multiplicative stretch $\alpha = 3$), the Thorup–Zwick distance oracle works as follows: landmarks are chosen independently at random with probability $p := 1/\sqrt{n}$ (see [Roditty et al. 2005] for deterministic landmark selection). Storing stars from all $np$ landmarks (in expectation) requires space proportional to $n^2p$. Node pairs at short distances are handled by precomputing and storing shortest-path trees and distances for open balls. Each node $u$ (with nearest landmark $l_u$) stores distances to all nodes $v$ within distance strictly less than $d(u, l_u)$, i.e., to all the nodes in the open ball $B(u) = \{v \in V : d(u, v) < d(u, l_u)\}$. The expected ball size is $1/p$, which yields expected total space requirements of $n^2p + n/p = O(n\sqrt{n})$.

Given a pair of nodes $(s, t)$ at query time, the algorithm checks whether $s \in B(t)$ or $t \in B(s)$, in which case the exact distance has been precomputed and can be returned. Otherwise, the error introduced by triangulation can be bounded using...
the triangle inequality: \(d(s,l_i) + d(l_i,t) \leq d(s,t) + d(t,l_i) + d(l_i,t) \leq 3d(s,t)\).

For \(k > 2\), landmarks are selected at \(k\) levels: at level \(i\), landmarks are sampled with probability \(p := n^{-i/k}\). Balls at higher levels are defined to contain only landmarks of the level below, hence balls at each level have expected size proportional to \(n^{1/k}\). The query algorithm alternates between landmarks of source and target. For details, see [Thorup and Zwick 2005; Wulff-Nilsen 2013].

Even Integral Stretch. Pătraşcu and Roditty [2010] (see also [Agarwal et al. 2011]) observe that the worst-case stretch is attained when neither \(s \in B(t)\) nor \(t \in B(s)\), but almost, meaning that the balls intersect, i.e. \(B(s) \cap B(t) \neq \emptyset\). They prove that, for unit-length graphs, there is a \((2,1)\)-approximate distance oracle using space \(O(n^{5/3})\). The tradeoff extends to weighted sparse graphs and to general \(k \geq 2\) with stretch \((2k - 2, 1)\) and space \(O(n^{1+2/2k-1})\) [Abraham and Gavoille 2011].

In the following, we briefly describe the simplified construction of Abraham and Gavoille [2011], building on [Patrascu and Roditty 2010]. Let the cluster of a node \(v\) contain all the nodes \(u\) that have \(v\) in their balls, i.e., \(C(v) := \{u : v \in B(u)\}\). An algorithm that carefully resamples landmarks can find a set of landmarks of size \(|L| = \tilde{O}(n^{2/3})\) such that, for all \(v \in V\), both \(|B(v)| = O(n^{1/3})\) and \(|C(v)| = \tilde{O}(n^{1/3})\) [Thorup and Zwick 2001]. Then, each node \(v\) stores distances to landmarks, to all nodes in its ball \(u \in B(v)\), and to all nodes whose ball has a non-empty intersection with \(B(v)\) (bounded by \(\tilde{O}(n^{2/3})\)).

The stretch \((2, 1)\) bound is proven as follows. If \(s \in B(t)\), \(t \in B(s)\), or \(B(s) \cap B(t) \neq \emptyset\), the exact distance has been precomputed and can be returned. Otherwise, since \(B(s) \cap B(t) = \emptyset\), and since the open ball \(B(u)\) has radius \(d(u,l_u) - 1\),

\[
(d(s,l_s) - 1) + (d(t,l_t) - 1) < d(s,t) \\
d(s,l_s) + d(t,l_t) \leq d(s,t) + 1.
\]

Without loss of generality, let us assume that the radius of \(B(t)\) is at most the radius of \(B(s)\), which is equivalent to \(d(t,l_t) \leq d(s,l_s)\). Therefore,

\[
d(t,l_t) \leq (d(s,t) + 1)/2.
\]

Then \(d(s,l_s) + d(l_s,t) \leq d(s,t) + 2d(l_s,t) \leq 2d(s,t) + 1\).

Sparse graphs. The Pătraşcu–Roditty result extends to graphs with general non-negative edge lengths and the additive term \(\beta = 1\) in the stretch can be avoided: for graphs on \(m\) edges, there is a distance oracle using space \(O(m^{1/3}n^{4/3})\) with stretch \(\alpha = 2\) and query time \(O(1)\) [Patrascu and Roditty 2010]. More generally, for many stretch values between \(\alpha = 2\) and \(\alpha = 3\), there is a distance oracle using space \(O(m^{1+1/(k+1/\ell)})\) with stretch \(\alpha = 2k - 1 \pm 2/\ell\) [Patrascu et al. 2012]. For an illustration of the tradeoff, see Fig. 1.

For multiplicative stretch \(\alpha < 2\), oracles with subquadratic space and constant query time are unlikely to exist (Section 2.1). However, for sparse graphs, oracles with subquadratic space and sublinear query time have been found [Fakcharoenphol and Saranurak 2010; Porat and Roditty 2013; Agarwal and Godfrey 2013]. Agarwal and Godfrey [2013] provide smooth tradeoffs between query time and stretch and also between query time and space. Further improvements have been announced [Agarwal 2013].
Fig. 1. Distance oracles (with $O(1)$ query time) for sparse graphs ($n$ nodes, $m = \tilde{O}(n)$ edges): the tradeoff between stretch [$\alpha$] and space [$S$], depicted using a linear scale for the stretch and a logarithmic scale for the space. Oracles with odd integral stretch (white circles) are due to Thorup and Zwick [2005]. Oracles with even integral stretch (gray circles) are due to Pătraşcu and Roditty [2010] and Abraham and Gavoille [2011]. Oracles in between (black dots) are due to Pătraşcu, Roditty, and Thorup [2012]. Their results say that, for many stretch values $\alpha \geq 2$, there is a distance oracle using space $S = O(n^{1+2/(\alpha+1)})$.

The oracle of Thorup and Zwick [2005] achieves $\tilde{O}(n)$ space for $k = \lg n$ with $O(\lg n)$ multiplicative stretch and $O(\lg n)$ query time. The oracle of Chechik [2013] improves the query time to $O(1)$. It would be interesting to reduce the stretch to $O(1)$ instead. The girth-based space lower bound proves that this is impossible for dense graphs. It is an open question whether such oracles exist for sparse graphs (Agarwal et al. [2011] refer to such oracles as the holy grail).

**Exact Distances.** An exact distance oracle for general graphs (which also works for directed graphs) with a different type of worst-case guarantee on the space and query complexities is due to Cohen, Halperin, Kaplan, and Zwick [2003]. Their 2-hop cover is a distance labeling scheme, which works as follows. Each node $u$ precomputes and stores distances to a set of landmarks $L(u)$ such that, for any pair of nodes $s,t$, at least one node on a shortest $s-t$ path is in $L(s) \cap L(t)$ (each shortest path is covered by a landmark). The query algorithm simply returns the best distance using a landmark $l \in L(s) \cap L(t)$. There is no absolute guarantee on the size of $L(\cdot)$, however, their polynomial-time preprocessing algorithm returns an $O(\lg n)$-approximation for the cover with minimum total size (cf. SETCOVER for the set of all shortest paths). Babenko, Goldberg, Gupta, and Nagarajan [2013] provide an $O(\lg n)$-approximation algorithm for the cover minimizing the maximum label size (and more general objective functions), thereby minimizing the worst-case query time (up to a log factor). 2-hop covers have been implemented, engineered, and evaluated for road networks (see Section 3.2.2) and also for more complex networks (see Section 4).
3. THEORY AND PRACTICE OF ROUTE PLANNING FOR ROAD NETWORKS

Efficiently finding “good” routes in transportation networks is arguably the main application scenario for shortest-path query methods. Due to its immediate practical implications, this scenario stimulated a large body of research.

As mentioned in the introduction, we assume that all relevant information is incorporated into the network and the length function. In the following, we shall not attempt to cover various modeling aspects, despite their practical importance. Let us just briefly mention that some methods discussed in this section (e.g. [Holzer et al. 2008; Delling et al. 2009; Delling et al. 2013a]) can in addition to mere expected travel times also effectively incorporate selected aspects of real-world road networks such as times spent at intersections and turn restrictions, route complexities (as, e.g. the number of turns), various uncertainties, fuel costs, and dynamic traffic information or time dependencies derived from historical traffic data. Most methods in this section, however, may require substantial modifications to fully incorporate such cost models.

We first give a brief historical overview and then we survey exact and approximate methods for planar graphs and methods for road networks, further subdividing into hierarchical and graph-labeling approaches.

Researchers started investigating point-to-point shortest path problems immediately after the introduction of the general shortest-path problem [Minty 1957; Dantzig 1960; Klee 1964; Smolleck 1975]. Experimental evaluation [Hitchner 1968; Bourgoin and Heurgon 1969; Dreyfus 1969; Gilsinn and Witzgall 1973; Pape 1974; Golden 1976; Cherkassky et al. 1996; Zhan and Noon 1998; Demetrescu et al. 2008] has always been a central part of research on shortest-path algorithms. Starting from classical single-source shortest-path algorithms it has been noted that, if an application requires only point-to-point distances, many SSSP algorithms can be stopped early. Furthermore, SSSP algorithms may run faster when executed from the source and the target simultaneously — this technique is also called bidirectional search [Dantzig 1963; Nicholson 1966; Boothroyd 1967; Chartres 1967; Murchland 1967; Pohl 1971]. Bidirectional search can be a very powerful technique for networks other than transportation networks as well.

Researchers further found that the representation of a graph in memory greatly affects the performance of the algorithm. For sparse graphs, representing the graph by an adjacency list is quite efficient, sorting each list by starting nodes (this representation is sometimes termed forward star form [Mehlhorn and Sanders 2008]). It may be efficient to also sort the edges of a node by their length [Dial et al. 1979]. Such a sorting step can be seen as preprocessing the graph in order to speed up the query algorithm (albeit without decreasing the worst-case query time).

Reordering nodes and edges was just the beginning. Researchers tried to further speed up the shortest-path algorithms of Dijkstra [1959], D’Esopo [Pollack and Wiebenson 1960; Pape 1974], and Moore [1959]. Network decomposition [Kitamura and Yamazaki 1965; Mills 1966; Land and Stairs 1967; Farbey et al. 1967; Hu 1968; Hu and Torres 1969] was used to speed up APSP algorithms on sparse networks (see Fig. 2). Other than the articles on the network decomposition technique, to the best of my knowledge, the thesis of Smolleck [1975] (see also [Smolleck and Chen 1981]) and an article of van Vliet [1978] appear to be among the first reports on the

shortest-path query problem with considerable preprocessing. Somewhat related, a method of Gallo [1980] effectively uses a prior shortest-path computation to speed up subsequent shortest-path computations (cf. [Klein 2005]). If, however, the next computation is “far” from the previous one, speedups may be minimal.

Fig. 2. Illustrations of early preprocessing techniques, extracted from the corresponding papers: Left: The network decomposition technique as originally depicted by Hu and Torres [1969]. First, the network is decomposed into overlapping subnetworks. Next, with each subnetwork treated separately, conditional shortest paths are obtained using triple operations. Finally, these conditional shortest paths are used to obtain the shortest paths between paired nodes in the original network by matrix mini-summation.

Right: The spider web transformations of van Vliet [1978], illustrated by contractions for nodes of degrees 2, 3, and 4. Van Vliet’s methods contract nodes such that groups of two or more links from the original network are combined into single links representing minimum distance paths between their end nodes.

Smolleck models the network by an electric circuit, wherein each edge is mapped to an impedance to efficiently answer approximate shortest-path queries. According to [Deo and Pang 1984], Smolleck achieves a speedup of 30 compared to Dijkstra’s algorithm (on a graph with 2,047 nodes and 2,547 edges); the paths are on average 1.9% longer than the optimal path; the preprocessing time is reported to be 1,000 times slower than the query time. Van Vliet introduced heuristics termed spider web techniques [van Vliet 1978, Section 6], which contract nodes and introduce shortcut edges (Fig. 2). Van Vliet in some sense combined APSP and SSSP techniques into a query method, illustrating the tradeoff that shortest-path

---

3Van Vliet attributes the idea to Hu [1969], who termed it distance-equivalent networks. There may be a connection to the minimum-route transformations of Akers [1960] and William S. Jewell (no reference). These network changes are based on Wye-Delta-transformations of electrical networks. However, the transformations appear to be restricted to planar networks and to two or three terminals. Hu and Torres [1969, p. 390] attribute smaller flow-equivalent networks to Akers [1960]. Van Vliet also relates it to triple operations [Floyd 1962; Murchland 1965; Hu 1968]. Such a triple operation compares an edge length with the lengths of paths with two edges using an intermediate pivot node. The method is mainly used in APSP algorithms.

query methods address. According to the article, van Vliet’s contraction techniques decrease the CPU time for multiple queries by approximately 25%.

For road networks, if in addition to the graph the geographical coordinates of the nodes are known, A* heuristics [Gelernter 1963; Samuel 1963; Doran 1967; Hart et al. 1968] based on the Euclidean distance have been used to guide the search towards the target [Sedgewick and Vitter 1986]. These heuristics are quite well known, rather easy to implement, and widely used. More recent techniques (see Section 3.2), however, yield substantially improved speedups.

3.1 Theoretical Results on Distance Oracles for Planar Graphs and Generalizations

Due to the importance of planar graphs as a more-or-less accurate model for road networks, shortest-path queries for planar graphs have been studied extensively. Real-world road networks may not actually be planar graphs but they seem to share some properties with planar graphs such as small separators and some sense of orientation [Eppstein and Goodrich 2008]. One might also argue that, among the graph classes theoreticians know how to design efficient algorithms for, planar graphs and their extensions are the closest to road networks. Other related graph classes with known approximate distance oracles are geometric graphs [Gudmundsson et al. 2008; Sankaranarayanan and Samet 2009; Sankaranarayanan et al. 2009], and graphs with bounded doubling dimension [Har-Peled and Mendel 2006; Abraham et al. 2008a; Bartal et al. 2011; Kawarabayashi et al. 2011].

3.1.1 Exact Shortest Paths. The contents of this section were partially extracted from [Mozes and Sommer 2012, Section 1.1]. In the following, as above, \(\tilde{O}(\cdot)\)–notation accounts for logarithmic factors. We give a brief overview of results; a summary can be found in Table III, illustrated in Fig. 3.

Djidjev [1996], improving upon Feuerstein and Marchetti-Spaccamela [1991], proves that, for any \(S \in [n, n^2]\), there is an exact distance oracle using space \(O(S)\) with query time \(O(n^2/S)\). Concurrent results for smaller ranges can be found in [Arikati et al. 1996; Buchholz and Riedhofer 1997; Riedhofer 1997]. These constructions use only recursive \(O(\sqrt{n})\)–separators [Ungar 1951; Lipton and Tarjan 1979; Djidjev 1985; Gilbert et al. 1984; Alon et al. 1990], and consequently, oracles with these space–query time tradeoffs also exist for bounded-genus and minor-free graphs. Experimental results indicate that real-world road networks appear to have recursive separators of size proportional to roughly \(\sqrt[3]{n}\) [Delling et al. 2011], except that road networks contain some grids of considerable sizes as subgraphs (with separators of size \(\Omega(k)\) for a \(k \times k\)–grid).

Djidjev’s method also follows the overall approach described in Section 1.2: the set of landmarks is chosen as the set of boundary nodes of an \(r\)–division [Friederickson 1987; Klein et al. 2013]. An \(r\)–division is essentially a partition of the edges into \(O(n/r)\) regions of size \(r\) such that each region \(R\) has at most \(O(\sqrt{r})\) boundary nodes \(\partial R\) (a node is called a boundary node if it is adjacent to edges in different regions). Next, pairwise distances among all landmarks are computed and stored. The space requirements for this distance table are \(S = O((n/\sqrt{r})^2)\). The query algorithm, given a pair of nodes \((s, t)\), first searches (using SSSP [Henzinger et al. 1997]) both regions \(R_s, R_t\). If \(s\) and \(t\) are in the same region \(R\), and if the shortest path is entirely contained in \(R\), the shortest-path distance has been found.
in time $O(r)$ (short-range query). Otherwise, exact distances to all corresponding landmarks (boundary nodes in $\partial R_s \times \partial R_t$, respectively) have been computed, and the distance is the minimum among all pairs of landmarks $(l_s, l_t) \in \partial R_s \times \partial R_t$ of $d(s, l_s) + d(l_s, l_t) + d(l_t, t)$. Since the number of boundary node pairs is bounded by $O((\sqrt{T})^2)$, the query time for long-range queries is also $O(r)$.

For a smaller range, also exploiting non-crossing properties of planar graphs, Djidjev proves that, for any $S \in [n^{4/3}, n^{3/2}]$, there is an exact distance oracle with space $O(S)$, and query time $\tilde{O}(n/\sqrt{S})$. For further improvements and extensions, see [Chen and Xu 2000; Cabello 2012; Nussbaum 2011; Mozes and Sommer 2012]. Djidjev observes that, if the boundary nodes $\partial R$ of each region $R$ form a simple cycle, then not all pairs $(l_s, l_t) \in \partial R_s \times \partial R_t$ need to be considered for long-range queries: the intersection pattern among shortest paths between nodes on two disjoint cycles of a planar graph is limited such that, instead of exploring $O((\sqrt{T})^2)$ pairs, the intermediate minimization step for long-range distances can be computed in time $O(\sqrt{T} \lg r)$ (the intersection pattern is restricted only if intermediate distances are computed in $G \setminus (R_s \cup R_t)$ and not in $G$). Distances from each node to its region’s boundary nodes are then either stored, computed at query time using an SSSP algorithm, or retrieved using multiple MSSP data structures [Klein 2005] (see also Section 3.1.3). Short-range distances can be computed faster by recursively computing distance oracles for each region.

<table>
<thead>
<tr>
<th>Preprocessing</th>
<th>Space</th>
<th>Query Time</th>
<th>Restriction (if any)</th>
<th>Reference(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td></td>
<td>[Henzinger et al. 1997]</td>
</tr>
<tr>
<td>$o(n^2)$</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td></td>
<td>[Wulff-Nilsen 2010]</td>
</tr>
<tr>
<td>$O(n^{n/2})$</td>
<td>$O(n^{n/2})$</td>
<td>$O(\sqrt{n})$</td>
<td>$S \in [n^{3/2}, n^3]$</td>
<td>[Arikati et al. 1996; Djidjev 1996]</td>
</tr>
<tr>
<td>$O(S)$</td>
<td>$O(S)$</td>
<td>$O(n^2/S)$</td>
<td>$S \in [n, n^{3/2}]$</td>
<td>[Djidjev 1996, Thm. 3]</td>
</tr>
<tr>
<td>$O(n^\sqrt{S})$</td>
<td>$O(S)$</td>
<td>$O(n^2/S)$</td>
<td>$S \in [n, n^{3/2}]$</td>
<td>[Djidjev 1996, Thm. 4]</td>
</tr>
<tr>
<td>$O(n \lg^2 n)$</td>
<td>$O(n \lg n)$</td>
<td>$O(n \sqrt{n} \lg^2 n)$</td>
<td></td>
<td>[Fakcharoenphol and Rao 2006; Klein et al. 2010]</td>
</tr>
<tr>
<td>$O(n \lg n)$</td>
<td>$O(n)$</td>
<td>$O(n^{3/2+\epsilon})$</td>
<td></td>
<td>[Nussbaum 2011; Mozes and Sommer 2012]</td>
</tr>
<tr>
<td>$O(n \lg \lg n)$</td>
<td>$O(n)$</td>
<td>$O(n/\poly(\lg n))$</td>
<td></td>
<td>[Italiano et al. 2011]</td>
</tr>
<tr>
<td>$O(n \sqrt{S})$</td>
<td>$O(S)$</td>
<td>$O(n/\sqrt{S} \lg n)$</td>
<td>$S \in [n^{4/3}, n^{3/2}]$</td>
<td>[Djidjev 1996, Thm. 5]</td>
</tr>
<tr>
<td>$O(n^2/S)$</td>
<td>$O(S)$</td>
<td>$O((S/n) \lg n)$</td>
<td>$S \in [n^{4/3}, n^{3/2}]$</td>
<td>[Chen and Xu 2000]</td>
</tr>
<tr>
<td>$O(n \sqrt{S})$</td>
<td>$O(S)$</td>
<td>$O((n/\sqrt{S}) \lg(n/\sqrt{S}))$</td>
<td>$S \in [n^{3/2}, n^2]$</td>
<td>[Chen and Xu 2000]</td>
</tr>
<tr>
<td>$O(S \lg n)$</td>
<td>$O(S)$</td>
<td>$O(n/\sqrt{S} \lg(n/\sqrt{S}))$</td>
<td>$S \in [n^{4/3}, n^2]$</td>
<td>[Nussbaum 2011, Thm. 4.1]</td>
</tr>
<tr>
<td>$O(S \lg n)$</td>
<td>$O(S)$</td>
<td>$O(n/\sqrt{S} \lg^{1.3} n)$</td>
<td>$S \in [n^{4/3} \lg^{1.3} n, n^2]$</td>
<td>[Cabello 2012, Thm. 12]</td>
</tr>
<tr>
<td>$O(S \sqrt{n} \lg^2 n)$</td>
<td>$O(S)$</td>
<td>$O(n/\sqrt{S})$</td>
<td>$S \in [n^{4/3}, n^3]$</td>
<td>[Nussbaum 2011, Thm. 5.2]</td>
</tr>
<tr>
<td>$O(S \sqrt{n} \lg n)$</td>
<td>$O(S)$</td>
<td>$O((n/\sqrt{S}) \lg^{2.5} n)$</td>
<td>$S \in [n \lg \lg n, n^7]$</td>
<td>[Mozes and Sommer 2012]</td>
</tr>
</tbody>
</table>

Table III. Time and space complexities of exact distance oracles for directed planar graphs. The tradeoff between space and query time is illustrated in Fig. 3. For the large-space result of Chen and Xu [2000], an additive inverse-Ackermann term in the query time is suppressed in this table.

Fakcharoenphol and Rao [2006] further exploit the non-crossing property (which they call the Monge property [Monge 1781; Hoffman 1963]). They call the complete bipartite graph among $\partial R_s \times \partial R_t$ in $G \setminus (R_s \cup R_t)$ a Dense Distance Graph (DDG). Their query algorithm can efficiently handle multiple DDGs simultaneously in time.

---

4More generally, $O(1)$ cycles can be handled; this simplified overview assumes a single boundary cycle per region.
Fig. 3. Distance oracles for planar graphs: The figure illustrates the tradeoff between the Space \( S \) and the Query time \( Q \) for different data structures on a doubly logarithmic scale, ignoring constant and logarithmic factors.

The upper line represents the \( Q = n^2 / S \) tradeoff (completely covered by [Djidjev 1996]; the range \( S \in [n^{3/2}, n^2] \) is covered by [Arikati et al. 1996], for \( S = n^{3/2} \) see also [Buchholz and Riedhofer 1997; Riedhofer 1997]; SSSP (\( S = Q = n \)) and APSP (\( S = n^2 \)) also lie on this line). Planarity is not necessary; only recursive separators of size \( O(\sqrt{n}) \) are assumed to achieve this tradeoff.

The lower line represents the \( Q = n / \sqrt{S} \) tradeoff; the range \( S \in [n^{4/3}, n^{3/2}] \) is covered by [Djidjev 1996]; extended to \( S \in [n^{4/3}, n^2] \) by [Chen and Xu 2000; Cabello 2012; Nussbaum 2011] (query time improvements by [Nussbaum 2011]), the point \( S = n \) is covered by [Fakcharoenphol and Rao 2006] (similar claims in [Buchholz 2000]), and the full range is covered by [Mozes and Sommer 2012].

roughly proportional only to the number of nodes in these DDGs (as opposed to the number of edges, which for DDGs is quadratic). Their technique is used for various distance oracles with low space and preprocessing complexities [Fakcharoenphol and Rao 2006; Italiano et al. 2011; Nussbaum 2011; Mozes and Sommer 2012].

The only lower bounds known are for distance labels, proving that total label length \( \Omega(n^{3/2}) \) is required for planar graphs with edge lengths [Gavoille et al. 2004, Corollary 3.11] (the best upper bound uses total label length \( O(n^{3/2} \log n) \) [Gavoille et al. 2004, Corollary 2.5]). There are no lower bounds on distance oracles for planar graphs. It is an open problem whether there exists another tradeoff curve strictly below \( Q = \tilde{O}(n / \sqrt{S}) \).

3.1.2 Approximations. To obtain constant or polylogarithmic query times while maintaining almost linear space, approximate distance oracles are considered. Thorup [2004] presents efficient \((1 + \epsilon, 0)\)-approximate distance oracles for directed planar graphs. One of the main ingredients of Thorup’s construction is a special separator consisting of a constant number of shortest paths (instead of a general set of \( O(\sqrt{n}) \) nodes as in the Lipton-Tarjan separator theorem [1979]). Each node computes and stores shortest-path distances to a set of \( O(1/\epsilon) \) landmarks per level, recursively for \( O(\log n) \) levels (see also [Klein and Subramanian 1998; Klein 2002] for related constructions). For directed graphs, the construction is actually more involved and the bounds show a moderate dependency on the largest edge length.
Distances among subsets of landmarks (those on the same shortest path \(Q\)) can be represented in a very compact way by just storing the position on \(Q\). Improved tradeoffs have been announced [Kawarabayashi et al. 2013]. Thorup's oracle for undirected graphs has been implemented and evaluated for road networks [Muller and Zachariasen 2007]. The results however indicate that, for these road networks, it is not competitive with the specialized methods discussed in Section 3.2.

Kawarabayashi, Klein, and Sommer [2011] extend Thorup's results to undirected graphs embedded in a surface of Euler genus \(g\). Abraham and Gavoille [2006] extend Thorup's result to minor-free graphs. They prove that minor-free graphs can be recursively separated using a (large) constant number of shortest paths. Based on these shortest-path separators, they then construct approximate distance oracles as in [Thorup 2004]. Kawarabayashi et al. [2011] provide tunable tradeoffs for the aforementioned approximate distance oracles for planar, bounded-genus, and minor-free graphs such that the space requirements can be made linear in the graph size while maintaining polylogarithmic query time (with techniques similar to those used for exact tradeoffs illustrated in Fig. 3).

3.1.3 Restricted queries.

Bounded-length queries. Kowalik and Kurowski [2006] prove that, for unit-length planar graphs, there is a distance oracle with linear preprocessing time and space requirements that answers queries for distances bounded by a constant \(h\) in constant time (which is an improvement over the \(O(\lg n)\) query time in Eppstein [1999, Theorem 12]). Dvorak, Král, and Thomas [2010] extend their result to essentially all sparse graphs (sparse as defined in [Nesetril and de Mendez 2006]).

One-face queries. Klein [2005] gives a distance oracle that preprocesses a graph with a specified face \(f\) in time and space \(O(n \lg n)\) to answer distance queries between any node incident to \(f\) and any other node (incident to an arbitrary face) in time \(O(\lg n)\). This data structure is also referred to as a Multiple Source Shortest Paths (MSSP) data structure and it is used as an ingredient in other distance oracles. Schmidt [1998] provides a similar data structure for grid graphs. Cabello and Chambers [2012] give a different and simpler algorithm, which also extends Klein’s result to graphs with genus \(g\).

3.2 Route Planning for Road Networks

Route planning for transportation networks (road networks in particular) has been studied intensively for many years. Recently, the 9th DIMACS Implementation Challenge, which took place in 2006, stimulated a lot of research with impressive results [Demetrescu et al. 2008]. In the following, we give a brief overview. The tradeoffs between space and query time are summarized in Fig. 4. For more details on recent results we refer to the survey on route planning [Delling et al. 2009a], the survey on A*-based point-to-point shortest-path queries [Goldberg 2007] (see also [Goldberg et al. 2009]), the overview on engineering large network applications [Zaroliagis 2008], and the Ph.D. theses of Schultes [2008] and Delling [2009]. Route planning is also strongly related to efficient path query processing on spatial networks [Papadias et al. 2003; Gupta et al. 2004; Demir et al. 2008; Samet et al. Submitted to: ACM Computing Surveys, Vol. V, No. N, September 2013.
Shortest-Path Queries in Static Networks

Fig. 4. Route planning for road networks: the tradeoff between space \( S \) and query time \( Q \) for recent shortest-path query data structures, depicted using doubly-logarithmic scales. The performance numbers represented by this figure were extracted from [Delling et al. 2009a, Table 1], [Bauer et al. 2010b, Table 8], [Abraham et al. 2011b, Table 1], [Arz et al. 2013, arXiv Table 5], and [Delling et al. 2013b, Table 2]. Performance numbers were obtained on different machines and scaled with best effort to make methods comparable. Colors and dashed lines do not carry any meaning; lines serve the purpose of visually connecting dots corresponding to different implementations or different variants of the same method.

Methods using Contraction Hierarchies (CH) [Geisberger et al. 2008; Sanders et al. 2008; Geisberger et al. 2012] dominate the low-space regime; methods based on Reach [Gutman 2004; Goldberg et al. 2009] and Highway Hierarchies (HH) [Sanders and Schultes 2005; 2006; Delling et al. 2009b] can be seen as the “first generation” of CH; Transit-Node Routing (TNR) [Bast et al. 2007a; Bauer et al. 2010b; Arz et al. 2013] and Hub Labels (HL) [Abraham et al. 2011b; Delling et al. 2013b] dominate the fast-query-time regime.

2008; Sankaranarayanan et al. 2009; Sankaranarayanan and Samet 2009]. Let us re-emphasize that the focus of this survey is on static networks. There are numerous methods that work with more dynamic transportation networks (for various definitions and interpretations of dynamic) but these methods are not considered in this survey.

From a technical perspective, two types of route planning methods can be distinguished: (i) methods that obtain improvements mainly by exploiting structural properties of the input graph (somewhat related to the exact methods for planar graphs, as described in Section 3.1.1), and (ii) methods that also exploit properties of the shortest-path metric (induced by the underlying edge lengths), which appears to be of hierarchical nature in road networks. Methods of that second type tend to yield significantly improved tradeoffs, at the cost of, potentially, being somewhat less “robust” (meaning that changes to the edge length function such as dynamic updates or incorporating realistic turn costs may have unexpected consequences to the methods’ performances).

Efficient practical methods to answer shortest-path queries are often devised by following a feedback loop that consists of four steps: design, analysis, implementation, and experimentation. This approach is also called algorithm engineering [Sanders 2009, Fig. 1]. Since experimentation is an integral part of the feedback

loop, the choice of the datasets may highly influence the outcome of the algorithm engineering process. Whenever possible, experiments are run with input graphs that are actually used in practice. Route planning methods discovered by an algorithm engineering process include, for example, Highway Hierarchies (HH) [Sanders and Schultes 2005; 2006] and its exceedingly popular successor called Contraction Hierarchies (CH) [Geisberger et al. 2008]. Both methods depend on structural properties of the input graph and rather heavily on the edge lengths and the shortest-path metric they impose. If the length function is chosen such that edge lengths correspond to Euclidean distances, the methods still work well but their performance is worse than the performance when edge lengths correspond to (estimated) travel times. It is for the so-called travel time metric, where these hierarchical methods excel, and where the performances obtained are truly impressive (see also other methods, as illustrated in Fig. 4). However, estimating travel times for road segments is a highly non-trivial task in itself and it is not entirely clear to what extent the estimates used in research datasets are accurate representations for actual travel times observed in the real world. To the best of my knowledge, there are only a few studies on the robustness of these methods, investigating whether the performances would drop significantly upon changes to the length function, see e.g. [Delling et al. 2013a]. Recent theoretical research (Section 3.2.4) strives to explain the success of these speedup techniques, analyzing the running times of preprocessing and query algorithms by appropriately modeling graph and metric properties of road networks.

In our overview of recent methods, two preprocessing strategies (mostly orthogonal to the above types) are distinguished. Approaches based on graph annotation attach additional information to each node or edge, based on which, at query time, the search tree can be prioritized or pruned. These approaches are inherently based on an SSSP algorithm such as Dijkstra’s algorithm. They are quite general in nature, and some also work very well on graphs other than those stemming from road networks. Hierarchical approaches are often somewhat more tailored towards their use in road networks. These algorithms usually compute an additional graph structure to speed up shortest-path queries.

3.2.1 Graph Annotation Approaches. An annotation approach is to attach additional information to nodes or edges of the graph. Based on this information, the query algorithm decides how to prioritize nodes in the queue, or which part of the graph not to search, i.e., how to prune the search space. A subset of these methods is sometimes also called goal-directed search algorithms.

A* [Gelernter 1963; Samuel 1963; Kung et al. 1986; Hart et al. 1968; Doran 1967] is a popular search technique in Artificial Intelligence. The idea is to direct the search towards the goal. In the priority queue implementation of Dijkstra’s algorithm, at each iteration, the node with the shortest distance to the source is extracted from the queue. In the A* algorithm, instead of ordering nodes by their distance from the source, nodes in the queue are ordered by their distance from the source plus a potential, which estimates the remaining distance to the target. By adding a potential to the priority of each node, the order in which nodes are removed from the priority queue is altered. A good potential function increases the priority of nodes that lie on a shortest path to the target (usually by decreasing
the priority of other nodes). In road networks, for example, if the coordinates of the target are known, the Euclidean distance provides a reasonable potential function [Sedgewick and Vitter 1986]. This has been exploited and applied successfully. In general, however, the coordinates may not be known. Metric embeddings or drawings [Wagner and Willhalm 2005] may provide coordinates.

Goldberg and Harrelson [2005] (see also [Goldberg and Werneck 2005; Goldberg et al. 2006; 2007]) propose to use a set of landmarks \( L \subseteq V \) and the triangle inequality to compute node potentials (their method is sometimes called ALT, short for A*, Landmarks, and Triangulation). Analogous to the distance oracle of Thorup and Zwick [2005], all nodes \( v \in V \) know the distance to all landmarks \( l \in L \). This auxiliary information fits perfectly into the framework of Section 1.2. For two nodes \( u, v \in V \) and a landmark \( l \in L \), the triangle inequality yields that \( d(u, v) \geq d(u, l) - d(v, l) \). Taking the maximum difference over all \( l \in L \) yields the best estimate, which is used as a potential in the A* search. The quality of the lower bound highly depends on the landmark selection. Since in the preprocessing phase the distances to all landmarks need to be computed and stored, the preprocessing time and the space consumption also depend on the number of landmarks. A central question is how to select few but good landmarks. Random selection is a straightforward approach but it may not necessarily provide good coverage, meaning that some nodes are far from all landmarks. Several heuristics have been proposed to improve coverage [Goldberg and Harrelson 2005; Goldberg and Werneck 2005], or to choose “important” nodes [Potamias et al. 2009]. Theoretical results on beacon-based triangulations [Kleinberg et al. 2009] characterize, to some extent, the strengths and weaknesses of ALT: for graphs with bounded doubling dimension, triangulation using a constant number of landmarks, yields \((1 + \epsilon, 0)\)-approximate distances for a \((1 - \sigma)\)-fraction of the nodes; it is also shown that this slack \( \sigma \) is necessary. While A* with landmarks works for general graphs, it can be expected to perform particularly well on graphs with low doubling dimension.

A* is easy to implement and it yields decent speedups. Bidirectional A*, however, is not entirely straightforward: either the termination criterion is changed [Pohl 1971; Kwa 1989], or the potential functions for the forward and the backward search need to be consistent; averaging the forward and backward potential yields a consistent potential function [Ikeda et al. 1994].

Precomputed Cluster Distances (PCD) [Maue et al. 2009] is a somewhat similar approach. The network is partitioned into clusters and distances between any pair of clusters are precomputed. These cluster distances yield upper and lower bounds for distances, based on which the search space can be prioritized or pruned.

Arc Flags (AF) [Lauther 2004; Köhler et al. 2005; Möhring et al. 2006]. The preprocessing algorithm partitions the graph into clusters and then, for each cluster \( C \), marks all edges where shortest paths towards nodes in \( C \) start. The query algorithm prunes edges that are not marked with the target cluster. A related approach uses geometric containers [Wagner and Willhalm 2003; Wagner et al. 2005]. On its own, AF preprocessing is rather expensive (there is a fast parallel preprocessing algorithm [Delling et al. 2013]). However, when applied within a hierarchy [Möhring et al. 2006] or when combined with other techniques, it can be very efficient [Bauer and Delling 2009; Bauer et al. 2010b].

Reach-Based Routing [Gutman 2004] is technically an annotation approach, however, it should, at least in spirit, also be considered a hierarchical approach. Reach is one of the first methods specifically capturing the hierarchical nature of road networks and exploiting it with provable correctness guarantees. Prior industrial methods classified roads according to heuristic hierarchies (often using road categories), thereby sacrificing correctness. In reach-based routing, each node is assigned a so-called reach value, which determines whether a particular node should be considered during Dijkstra’s algorithm. To have a high reach value, a node must lie on a shortest path that extends a long distance in both directions from the node. A node is excluded from consideration if its reach value is small, that is, if it does not contribute to any path long enough to be of use for the current query. When combined with shortcuts [Goldberg et al. 2009], Reach is rather similar to many hierarchical approaches.

3.2.2 Hierarchical Approaches. Hierarchical methods to compute shortest paths in graphs have been proposed by many researchers. Many methods effectively exploit the inherent hierarchical nature of road networks. However, in this section, hierarchical does not exclusively refer to this hierarchy of roads. Many methods construct an auxiliary graph with multiple levels: a hierarchy of graphs. A shortest-path query is then answered by searching only a small part of the auxiliary graph, often using Dijkstra’s algorithm. In the following, we give a brief overview of selected recent hierarchical approaches.

Multi-Level Overlay Graphs [Jung and Pramanik 1996; Jing et al. 1998; Schulz et al. 2002; Holzer et al. 2008; Delling et al. 2009; Delling et al. 2013a] build a hierarchy of graphs with node sets at higher levels chosen as subsets of the node sets at lower levels (cf. a hierarchy of landmarks). Two nodes \( u, v \) at level \( i \) may be connected by an edge with length corresponding to the distance in \( G \) if the shortest path in level \( i - 1 \) does not use any other node of level \( i \). Selecting the landmark set on higher levels is one of the most critical components of these methods; several selection heuristics are proposed and evaluated. Highway-Node Routing (HNR) [Schultes and Sanders 2007] effectively uses highway nodes as landmarks (see also paragraph below). Customizable Route Planning (CRP) [Delling et al. 2013a] uses small recursive separators [Delling et al. 2011]. CRP is currently used in Microsoft Bing Maps. See also Section 3.1.1 for similar separator-based methods for planar graphs. Previous methods based on separators were significantly less efficient; among other reasons, the performance of CRP is very good since the preprocessing algorithm puts substantial effort into minimizing the sizes of the separators [Delling et al. 2011].

Highway Hierarchies (HH) [Sanders and Schultes 2005; 2006] are based on the observation that a certain class of edges (the highway edges) tend to have greater representation among the portion of the shortest paths that are not in the vicinity of either the source or target (similar to high reach values [Gutman 2004]). A recursive computation of these edges, paired with a contraction step, leads to a hierarchy of graphs that enables an impressive speedup at query time.

Contraction Hierarchies (CH) [Geisberger et al. 2008; Geisberger et al. 2012] is the exceedingly popular successor of HH. An integral ingredient of HH is its initial contraction step. Nodes with low degree can be contracted, since their removal...
Shortest-Path Queries in Static Networks

does not cause many additional edges (an observation related to van Vliet’s spider web [van Vliet 1978] and Hu’s distance-equivalent networks [Hu 1969]). This observation can be generalized [Geisberger et al. 2008]: for each node, the number of potential shortcut edges is computed. If for a node under consideration the number of shortcuts is smaller than the number of expected shortcuts based on the node degree, the node is contracted. A node contraction can also be interpreted as a particularly structured way of adding shortcuts. Thinking about shortest-path queries in road networks, one almost immediately notes that many nodes have degree 2 (in the undirected sense) and that these can be contracted. Van Vliet [1978] contracts nodes up to degree 4; it is reported that contractions of nodes with higher degrees did not yield any speedup but a slowdown. CH uses intelligent heuristics to contract nodes in the “right” order. This order $\pi$ defines a directed star for each node as follows: node $u$ with rank $\pi(u)$ must be connected to a node $v$ with $\pi(v) > \pi(u)$ if the shortest path from $u$ to $v$ does not use any node $w \neq v$ with $\pi(w) > \pi(u)$. The union of all these directed stars defines the forward (or upward) CH. The backward CH is defined analogously. Using these compact auxiliary graphs, the bidirectional query algorithm can efficiently find shortest paths. Contraction-based techniques perform very well in practice, the space overhead is small, and the preprocessing step is particularly efficient.

Transit-Node Routing (TNR) [Bast et al. 2007a; Bast et al. 2007b; Arz et al. 2013] is based on the following observation: when driving somewhere sufficiently far away, drivers usually leave their current location via one of only a few access routes to a relatively small set of landmarks called transit nodes. These landmarks are then interconnected by a network relevant for long-distance travel. The TNR method precomputes all shortest paths to landmarks (stars) and all shortest paths among landmarks (clique). The preprocessing is quite expensive but the query time is very low, since, for any two locations far enough, it essentially requires only a few dozen table lookups for all pairs of corresponding landmarks. A recent variant of TNR [Arz et al. 2013] can be interpreted as a sophisticated combination of CH with a distance table: transit nodes are chosen as the top–$k$ nodes in contraction order, short-range queries are computed using CH, while long-range queries correspond to several table lookups. Depending on the needs of the application, the number of transit nodes $k$ can be varied, thereby determining the tradeoff between space and query time.

Hub Labels (HL) [Abraham et al. 2011b; 2012b; Delling et al. 2013b] are used in the method that currently offers the fastest query times. During preprocessing, each node $u$ computes and stores the distance to a set of carefully chosen landmarks $L(u)$ in its label (stars; for directed graphs, different landmarks $L^+(u)$, $L^-(u)$ may be used for forward and backward distances). At query time, given two labels corresponding to distances to landmarks $L(s)$, $L(t)$, respectively, the algorithm simply computes and outputs $\min_{l \in L(s) \cap L(t)} d(s,l) + d(l,t)$. This can even be implemented in SQL, which allows for more general queries involving, e.g., points of interest [Abraham et al. 2012a]. As long as $L(s)$, $L(t)$ are small, the query algorithm is efficient. Furthermore, if labels are stored consecutively for each node, the query algorithm also has good locality. The main difficulty lies in choosing $L(u)$ for a node $u$: for any two nodes $s$, $t$, the intersection of their landmark sets $L(s) \cap L(t)$

shall contain at least one node on a shortest $s - t$ path, i.e. the set of landmarks must cover all shortest paths (see [Cohen et al. 2003] and Section 2.2). For road networks, small labels can be computed efficiently using CH search spaces. In the HL method, shortest paths are represented by two hops; in the TNR method, three hops are used. The extremely fast query times are paid for by rather high space requirements (compression in [Delling et al. 2013b]).

3.2.3 Combinations. Combining graph annotation and hierarchical approaches often yields powerful methods. Several combinations have been investigated and evaluated empirically [Holzer et al. 2005; Bauer et al. 2010b]. Particularly strong combinations are Reach with Shortcuts [Goldberg et al. 2009], CHASE [Bauer et al. 2010b], which combines Contraction Hierarchies with Arc Flags, and SHARC [Bauer and Delling 2009; Brunel et al. 2010], which combines Shortcuts with Arc Flags.

To sum up, the “tricks of the trade” [Bast 2009, Section 3] for fast routing on transportation networks are bidirectional search, exploiting hierarchy, graph contraction, goal direction, and distance tables. Let us conclude by noting that some of the methods described in this section, despite being tailored towards their use in road networks, can be adapted to work on other networks as well, such as those stemming from public transportation, albeit with somewhat reduced speedups [Berger et al. 2009; Bast 2009].

3.2.4 Analysis. The observed performance of the aforementioned methods is outstanding, however, complexity results are mostly experimental (exactness and correctness are proven).

There are some worst-case results for various speedup techniques. One core part of many speedup techniques, particularly the hierarchical ones, is the insertion of shortcuts; a shortcut is an additional edge $(u, v)$ whose length is equal to the distance from $u$ to $v$, and that represents shortest $u - v$-paths in the graph. Let the hop-length of a path be defined as the number of edges on a shortest path. The shortcut problem [Bauer et al. 2009] consists of adding a fixed number of shortcuts to a graph such that the sum of the hop lengths of hop-minimal shortest paths on the graph is minimized. This optimization problem is difficult to solve both optimally and approximately unless $\text{P} = \text{NP}$ [Bauer et al. 2009]. If the shortest paths are unique, a greedy algorithm can find a solution that is optimal up to a constant factor.

Contraction Hierarchies can be seen as a structured way of adding shortcuts by contracting nodes in a sophisticated order [Geisberger et al. 2008; Geisberger et al. 2012b]. However, computing or even approximating the optimal ordering is NP-hard [Bauer et al. 2010a]. For graphs with small recursive separators such as planar graphs there are bounds on CH preprocessing and space [Milosavljevic 2012] as well as query time [Bauer et al. 2013] (some based on a relation to nested dissection [Lipton et al. 1979]).

Abraham et al. [2010; 2011a] found that, if a graph has low highway dimension, algorithms based on Reach [Gutman 2004; Goldberg et al. 2009], Contraction Hierarchies [Geisberger et al. 2008], Hub Labels [Abraham et al. 2011b], and SHARC [Bauer and Delling 2009] have provable efficiency guarantees. Intuitively, a graph has small highway dimension if, for every radius $r > 0$, there is a sparse
set of nodes $S_r$ such that every shortest path of length greater than $r$ includes a node from $S_r$. A set is deemed sparse if every ball of radius $O(r)$ contains only a small number of elements of $S_r$. Computing and analyzing the highway dimension of real road networks remains an open problem.

4. COMPLEX NETWORKS

Recently, shortest-path query algorithms and data structures have been studied for more general graphs, motivated by potential applications for real-world networks such as social networks or regulatory networks from biology. This section of the survey is rather vague, since research on shortest-path queries for complex networks seems to currently be evolving quite rapidly. Also, the absence of commonly agreed benchmarks poses difficulties on evaluation and comparison of existing algorithms. Similar difficulties arise for theoretical work, where there is currently a rather large variety of random-graph models for complex networks. For most of these complex network models, the following common properties have been identified: (i) complex networks appear to have small diameters (proportional to roughly $\lg n$; this property is referred to as the small-world property), (ii) oftentimes there is a large variety of node degrees (scale-free networks; the degree sequence obeys a power law), and (iii) there seem to be no small separators (linear-sized core). Most of these properties currently cannot be exploited by common algorithmic techniques; other properties such as the one that these networks have nodes with high degree (so-called hubs) have been exploited successfully in (approximate) shortest-path query algorithms.

Most results on shortest-path queries in complex networks are of experimental nature. I am aware of only few results with worst-case bounds, which are the following compact routing schemes. Enachescu et al. [2008] analyze the compact routing scheme of Thorup and Zwick [2001] for $G_{n,p}$ random graphs. They prove that stretch $\alpha = 2$ can be achieved with space $O(n^{7/4})$ by selecting $O(n^{3/4})$ landmarks (later dominated by the $O(n^{5/3})$-space oracle of Pătraşcu and Roditty [2010] for general graphs). They also claim (without proof in the proceedings version) that multiplicative stretch $\alpha$ can be achieved with space $O(n^{1+2/(\alpha+1)+\epsilon})$. See also [Krioukov et al. 2004] for results on the stretch distribution. Chen et al. [2012] provide an approximate distance oracle with stretch $\alpha = 3$ using space $O(n^{4/3})$ for certain random power-law graphs [Aiello et al. 2000; Chung and Lu 2002] (the actual space requirements may be smaller, depending on the power-law exponent). Their oracle is an adaptation of the Thorup-Zwick distance oracle using high-degree nodes as landmarks. Given that shortest paths in complex networks are usually very short (small worlds, six degrees of separation), it is however questionable whether worst-case guarantees on the multiplicative stretch of $\alpha = 2, 3$ are particularly useful.


---

5 Whether or not many of these degree sequences actually obey power laws is a controversial question [Faloutsos et al. 1999; Clauset et al. 2009; Achlioptas et al. 2009; Roughan et al. 2011].

6 Any compact routing scheme may serve as an approximate shortest-path oracle, retrieving each edge of the path by simulating the decision of each router. We do not attempt to cover results on compact routing in this review, instead referring to [Gavoille and Peleg 2003; Thorup and Zwick 2001; Abraham et al. 2008b] and the references therein.

and random power-law graphs. The Thorup-Zwick distance oracle uses information precomputed in two steps (see also Section 2.2). The query result is either (i) a local exact distance or (ii) a triangulation via landmarks. Many implementations focus on the triangulation part, providing good estimates for long-range distances by carefully selecting landmarks [Potamias et al. 2009; Das Sarma et al. 2010; Gubichev et al. 2010; Tretyakov et al. 2011; Cao et al. 2011; Qiao et al. 2011; Qiao et al. 2012; Cheng et al. 2012]. Agarwal et al. [2012] focus on the local part, computing distances using ball intersections (which they call vicinities) with landmarks sampled with probability proportional to their degrees. For corresponding worst-case bounds, see also [Patrascu and Roditty 2010; Agarwal et al. 2011].


**Core–Fringe Methods.** In the stretch–(1, D) routing scheme of Brady and Cowen [2006], the algorithm first computes a shortest-path tree from the node with the highest degree. All nodes up to distance D/2 for some parameter D form the core with diameter D. The remaining nodes form the fringe, which is claimed to be almost a forest. At query time, distance estimates are computed as the minimum among the distances in multiple trees. Wei [2010] explores the property that many scale-free networks have reasonably small tree-width outside the core (see also [Akiba et al. 2012] for further speedups).


5. SUMMARY

Shortest-path query processing in graphs has been studied extensively both by theoreticians and practitioners. Practical investigations focus mainly on the class of transportation networks, for which substantial speedups with respect to classical SSSP algorithms can be achieved. For transportation networks, the focus of practical research efforts appears to be shifting towards dynamic and personalized scenarios. For complex networks, methods have been proposed only recently; their efficiency and optimality is still under investigation. Common benchmark networks have not crystallized yet.

Recent theoretical research on distance oracles for general graphs has been centered around improving preprocessing and query times (due to restrictive space lower bounds). For restricted graph classes such as sparse graphs, planar graphs, and power-law graphs, various questions remain to be solved. Distance oracles for directed graphs of restricted classes are mostly unknown territory.
ACKNOWLEDGMENTS

Many thanks to Michael E. Houle for his encouragement in writing this survey and for his valuable comments on an earlier version [Sommer 2010, Chapter 3]. I learned almost everything included in this survey from great collaborators while working on [Sommer et al. 2009; Honiden et al. 2010; Djidjev and Sommer 2011; Gavoille and Sommer 2011; Kawarabayashi et al. 2011; Akiba et al. 2012; Lim et al. 2012; Chen et al. 2012; Mozes and Sommer 2012; Kawarabayashi et al. 2013]. This survey has improved substantially thanks to many comments and suggestions from Daniel Delling, Ron Gutman, Shay Mozes, Dirck van Vliet, Christian Wulff-Nilsen, and the anonymous referees of this journal.

REFERENCES

Agarwal, R., Caesar, M., Godfrey, P. B., and Zhao, B. Y. 2012. Shortest paths in less than a millisecond. In 5th Workshop on Online Social Networks (WOSN).


Karlin, Y. 1929. *Lanczsems*.


Shortest-Path Queries in Static Networks


Roughan, M., Willinger, W., Maennel, O., Perouli, D., and Bush, R. 2011. 10 lessons from 10 years of measuring and modeling the Internet’s autonomous systems. IEEE Journal on Selected Areas in Communications 29, 9, 1810–1821.


