More Compact Oracles for Approximate Distances in Planar Graphs

Christian Sommer
csom@mit.edu

November 1, 2011

Abstract

Distance oracles are data structures that provide fast (possibly approximate) answers to shortest-path and distance queries in graphs. The tradeoff between the space requirements and the query time of distance oracles is of particular interest and the main focus of this paper.

In FOCS'01, Thorup introduced approximate distance oracles for planar graphs. He proved that, for any $\epsilon > 0$ and for any planar graph on n nodes, there exists a $(1+\epsilon)$–approximate distance oracle using space $O(n\epsilon^{-1}\log n)$ such that approximate distance queries can be answered in time $O(\epsilon^{-1})$.

Ten years later, we give the first improvements on the space–query time tradeoff for planar graphs.

- We give the first oracle having a space–time product with subquadratic dependency on $1/\epsilon$. For space $\tilde{O}(n\log n)$ we obtain query time $\tilde{O}(\epsilon^{-1})$ (assuming polynomial edge weights). We believe that the dependency on ϵ may be almost optimal.
- For the case of *moderate* edge weights (average bounded by $poly(\log n)$, which appears to be the case for many real-world road networks), we hit a "sweet spot," improving upon Thorup's oracle both in terms of ϵ and n. Our oracle uses space $\tilde{O}(n\log\log n)$ and it has query time $\tilde{O}(\epsilon^{-1}+\log\log\log n)$.

(Notation: $\tilde{O}(\cdot)$ hides low-degree polynomials in $\log(1/\epsilon)$ and $\log^*(n)$.)

1 Introduction

Distance oracles [TZ05] generalize the all-pairs shortest paths problem as follows: instead of computing and storing a distance matrix (with quadratic space requirements and pairwise distance computations that require *one* table look-up only), we wish to compute a data structure that requires sub-quadratic space S but still allows for *efficient* (as in sublinear query time Q) distance computations. Depending on the application, it may be acceptable to output *approximate* answers to shortest-path and distance queries. The estimate provided by the distance oracle is supposed to be *at least* as large as the actual distance. The *stretch* $\alpha \geq 1$ of an approximate distance oracle is defined as the worst-case ratio over all pairs of nodes of the query result divided by the actual shortest-path length.

Distance oracles can potentially be used in applications such as route planning and navigation [Gol07, Zar08, DSSW09], Geographic Information Systems (GIS) and intelligent transportation systems [JHR96], logistics, traffic simulations [ZKM97, BBJ+02, RN04, BG07], computer games [Sto99], server selection [NZ02, DCKM04, CCRK04, ST08, EBN09], XML indexing [STW04, STW05], reachability in object databases, packet routing [SS80], causal regulatory networks [CZE+11], and path finding in social networks [Kar29, Mil67, New01].

For general graphs, distance oracles use large amounts of space, or they have long query time, or their stretch is at least two [TZ05, SVY09, PR10]. In this work we consider planar graphs, for which the known tradeoffs between stretch, space, and query time are much better (see Table 1 for an overview).

One important reason for this better tradeoff performance is that algorithms can make use of small *separators* [Ung51, LT79, Mil86]. For approximate distance oracles, separators consisting itself of a small number of shortest paths are particularly useful [Tho04, Kle02]. Separator-based approaches however often use recursion of logarithmic depth, which manifests itself by logarithmic factors in either the space requirement or the query time (or even both). Currently, the best tradeoff is provided by Thorup's approximate distance oracle, for which we provide a brief technical outline in Sections 2.1 and 2.2. The best exact distance oracles [FR06, Dji96, CX00, Cab06, Nus11, MS12] have a space-query time product $S \cdot Q$ proportional to roughly $n\sqrt{n}$. For the best $(1+\epsilon)$–approximate distance oracle [Tho04], that product $S \cdot Q$ is $O(n \log n)$ for constant ϵ.

Space	Query	Stretch	Reference
$O(n)$	$O(n)$	1	SSSP [HKRS97]
$O(n^2)$	$O(1)$	1	APSP
$O(n^2 \log\log(n)/\log(n))$	$O(1)$	1	[WN10a]
$O(S)$	$O(nS^{-1/2}\log^2(n)(\log\log n)^{3/2})$	1	[MS12] for any $S \in [n\log\log n, n^2]$
$O(n \log n)$	$O(\sqrt{n}\log^{5/2} n)$	1	[FR06]
$O(n)$	$O(n^{(1/2)+\delta})$	1	[MS12] for any constant $\delta > 0$
$O(n\epsilon^{-1}\log n)$	$O(\epsilon^{-1})$	$1+\epsilon$	[Tho04]
$O(n)$	$O(\epsilon^{-2}\log^2 n)$	$1+\epsilon$	[KKS11]
$\tilde{O}(n \log n)$	$\tilde{O}(\epsilon^{-1})$	$1+\epsilon$	Theorem 1 (up to $\log(1/\epsilon)$ and $\log^*(n)$)
$\tilde{O}(n \log\log n)$	$\tilde{O}(\epsilon^{-1} + \log\log\log n)$	$1+\epsilon$	Theorem 2 (up to $\log(1/\epsilon)$ and $\log^*(n)$)
$O(n \log\log n)$	$O(\log\log\log n)$	$O(1)$	Proposition 1

Table 1: Time and space complexities of distance oracles for undirected planar graphs (some results extend to planar digraphs). Existing results are for arbitrarily weighted graphs; our new results are for graphs with polynomial weights (Theorem 1) and for *moderately weighted* graphs (Theorem 2) only. $\log^*(n)$ is the *iterated logarithm* of n. For the bounds corresponding to our results, $\tilde{O}(\cdot)$ hides low-degree polynomials in $\log(1/\epsilon)$ and $\log^*(n)$.

Our main result is a distance oracle that improves upon these tradeoffs in terms of *both* n *and* ϵ for graphs with *moderate* edge weights. By moderate we mean that, after normalization such that the smallest edge

weight is 1, the average weight is bounded by $poly(\log n)$. We believe that this is a reasonable assumption as the average weight for the European road network (the version made available for scientific use by the company PTV AG) appears to be not too large.[1] We provide a $(1+\epsilon)$–approximate distance oracle with the following characteristics: for constant ϵ, the space is $S = O(n \log\log(n) \log^*(n))$ and the query time is $Q = O(\log\log\log n)$, thus $S \cdot Q = o(n \log n)$. We show how the logarithmic dependency on n, which is quite common in separator-based approaches, can be avoided entirely, which may be interesting in its own right.[2] Equally important, our construction also yields the first space–time product with *subquadratic* dependency on ϵ (known constructions have ϵ^{-1} in both space and query [Tho04, Kle02] or ϵ^{-2} in the query complexity [KKS11]).

More precisely, we prove the following. All our bounds are in the *word RAM model* [CR73].

Theorem 1. *For any undirected planar graph G on n nodes with edge weights polynomial in n and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n))$ using space $O(n\,[\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$, and preprocessing time $O(n\epsilon^{-2} \log^4(n) \log^*(n) + \mathcal{SNC}(n) \cdot \log n)$, where $\mathcal{SNC}(n)$ denotes the time required to compute a sparse neighborhood cover.*

Theorem 2. *For any undirected planar graph G on n nodes with average weight $\leq \log^\theta n$ for some constant $\theta \geq 0$, and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n) + \log\log\log(n))$ using space $O(n\,[\log\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$, and preprocessing time $O(n\epsilon^{-2} \log^3(n) \log\log(n) \log^*(n) + \mathcal{SNC}(n) \cdot \log\log n)$, where $\mathcal{SNC}(n)$ denotes the time required to compute a sparse neighborhood cover.*

As a starting point, we provide the following distance oracle with constant stretch.

Proposition 1. *For any undirected planar graph G on n nodes with average weight bounded by $\log^\theta n$ for some constant $\theta \geq 0$ there exists an $O(1)$–approximate distance oracle with query time $O(\log\log\log n)$ using space $O(n \log\log n)$, and preprocessing time $O(n \log^3 n + \mathcal{SNC}(n) \cdot \log\log n)$, where $\mathcal{SNC}(n)$ denotes the time required to compute a sparse neighborhood cover.*

Somewhat surprisingly, nothing more efficient than Thorup's $(1+\epsilon)$–stretch oracle had been known for arbitrary constant approximations. The only $O(1)$–stretch (as opposed to $(1+\epsilon)$–stretch) oracle constructions for planar graphs we are aware of are [Che95, ACC+96, GKR01] and the following two indirect constructions: one construction is by using a routing scheme [FJ89, FJ90] and another construction is by using an ℓ_∞–embedding [KLMN04]. The space–query time tradeoff of our data structure is better than that of all the previous constructions.

Techniques Our data structures make extensive use of Thorup's approximate distance oracle [Tho04] and of a large variety of techniques developed for planar graphs in recent years. We also introduce a new kind of *r–divisions* [Fre87] based on shortest-path separators. Other techniques we use include *crossing substitutes* [KS98], *sparse neighborhood covers* [BLT07], generalized *dominating sets* [KP98], and the *Cycle MSSP* data structure [MS12].

2 Preliminaries

We use standard terminology from graph theory, see for example [Die05]. Graphs we consider are planar, undirected, unweighted (if not explicitly mentioned otherwise — the extension to moderately weighted graphs

[1] The network covers European 14 countries and it has 18,010,173 nodes and 42,560,279 edges. It serves as an important benchmark graph for shortest-path query methods [DGJ08]. For the travel time metric (`scc-eur2time.gr`), the total weight is 21,340,824,356, which yields an average of approximately 501. For the distance metric (`scc-eur2dist.gr`) the two values are 9,420,195,951 (sum) and 221 (rounded average). Also, $\log_2(42560279) \approx 25.34$ and thus $(\log_2 |E|)^2 \geq 501$. We do not claim to rigorously distinguish $O(poly(\log|E|))$ from $\Omega(\sqrt{|E|})$ for this E (here $\sqrt{42560279} \approx 6523.82$).

[2] Recently, the complexity of some optimization problems for planar graphs has been improved [WN10b] from worst-case running time $O(n \log n)$ to $O(n \log\log n)$ using fast r–*divisions* [Fre87] (and other techniques). For approximate distance oracles, a similar approach improves the space requirements at the cost of the query time [KKS11]. In this work, we improve the *tradeoff*, i.e. the product of space and query time.

is emphasized in the pseudocode), and they have n nodes.

Let $\log^*(\cdot)$ denote the *iterated logarithm* function (also *super-logarithm*), which is defined as $\log^* n = 1 + \log^*(\log n)$ for $n > 1$ and as 0 for $n \leq 1$.

2.1 Planar Separators and r–divisions

A *separator* for a graph $G = (V, E)$ is a subset of the nodes $S \subseteq V$ such that removing the nodes in S from G partitions the graph into at least two disconnected components. Let us assign a *weight* $w \in [0, 1]$ to every node $v \in V$. A separator is deemed *balanced* if none of the resulting components has weight more than a constant fraction ρ of the total weight for some constant $\rho < 1$.

Planar graphs are known to have *small* separators: for any planar graph there exists a balanced separator consisting of $O(\sqrt{n})$ nodes [Ung51, LT79, Mil86]. Recursively separating a graph into smaller components, we obtain a *division* into edge-induced subgraphs. A node of G is a *boundary node* of the partition if it belongs to more than one subgraph. An *r–division* [Fre87] partitions G into $O(n/r)$ subgraphs, called *regions*, each consisting of $O(r)$ edges with at most $O(\sqrt{r})$ nodes on the boundary.

Separators can be chosen, for example, to form a cycle [Mil86] or also to form a set of paths [LT79, Tho04]. Lipton and Tarjan [LT79] prove that, for any spanning tree T in a triangulated planar graph, there is a non-tree edge e such that the unique simple cycle in $T \cup \{e\}$ is a balanced separator (fundamental cycle separator). Thorup [Tho04] uses this construction with a *shortest-path tree* T, rooted at an arbitrary source node. For any node u, let $T(u)$ denote the tree path from u to the root.

Lemma 1 (Shortest-Path Separability; Thorup [Tho04, Lemma 2.3])**.** *In linear time, given an undirected planar graph H with a rooted spanning tree T and non-negative vertex weights, we can find three vertices u, v, and w such that each component of $H \setminus V(T(u) \cup T(v) \cup T(w))$ has at most half the weight of H.*

Since T is a shortest-path tree, a separator S consists of at most three *shortest* paths.

2.2 Approximate Distance Oracle and Labeling Scheme

The approximate distance oracle for planar graphs by Thorup [Tho04] can be distributed as a distance labeling scheme [Pel00, GKK+01, GPPR04]. Each node u is assigned a label $\mathcal{L}(u)$ such that there is a decoding function $\mathcal{D}(\cdot, \cdot)$ that approximates the distance between u and v based on their labels only.

Lemma 2 (Thorup [Tho04, Theorem 3.16 and Theorem 3.19])**.** *There is an algorithm that computes $(1+\epsilon)$–approximate distance labels for an n–node planar graph with the following properties. The algorithm runs in time $O(n\epsilon^{-2} \log^3 n)$ and outputs labels of length $O(\epsilon^{-1} \log n)$ with query time $O(\epsilon^{-1})$.*

One of the key ideas in Thorup's oracle is the concept of *shortest-path tree separability* as outlined in the previous section (Lemma 1). A constant number of shortest paths Q separate the graph into components of at most half the size. Another key idea is to approximate shortest $s - t$ paths that intersect a separator path Q. Let the shortest $s - t$ path intersect Q at a particular node $q \in Q$. If we are willing to accept a slightly longer path, we can restrict the number of possible intersections from $|Q|$ to $O(1/\epsilon)$, for $\epsilon > 0$. We use the following variant of Klein and Subramanian [KS98] (which they also call a *crossing substitute*).

Lemma 3 (Klein and Subramanian [KS98, Lemma 4])**.** *Let Q be a path of length $O(D)$ for an integer D. Let $C \subseteq Q$ be a set of $O(1/\epsilon)$ equally spaced nodes on Q, called* the ϵ–cover of Q. *Any pair of nodes (u, v) at distance $[D, 2D)$ whose shortest path intersects Q can be* $(1 + \epsilon)$–approximated *by a path going through a node $c \in C$, that is,*

$$d(u, v) \leq \min_{c \in \mathcal{C}} d(u, c) + d(c, v) \leq (1 + \epsilon) d(u, v).$$

The proof is based on the observation that any *detour* using node $c \in C$ instead of node q could cause an additive error of at most $O(\epsilon D)$.

2.3 Distance–δ Dominating Sets

Let $\delta < n$ be an integer. A *δ–dominating set* of a graph $G = (V, E)$ is a subset $L \subseteq V$ of nodes such that for each $v \in V$ there is a node $l \in L$ at distance at most δ. It is well-known that there is a δ–dominating set L of size at most $|L| \leq n/(\delta + 1)$ and that such a set L can be found efficiently [KP98].

A related but hard (even for degree–3 planar graphs) problem is the *minimum p–center problem* [Ple80, DF85, Ple87], where we want to compute a set $U \subseteq V$ of size $p = |U|$ such that the maximum distance from any node $v \in V$ to U is minimized. We are mainly interested in that worst-case distance, for which the best guarantee is n/p.

2.4 Sparse Neighborhood Covers

Busch, LaFortune, and Tirthapura [BLT07] provide *sparse covers* [AP90, ABCP98, AGMW07, BLT07] for planar graphs.

Lemma 4 (Busch, LaFortune, and Tirthapura [BLT07])**.** *For any planar graph G and for any integer r, there is a* sparse cover*, which is a collection of connected subgraphs $(G_1, G_2, \dots)$, with the following properties:*

- *for each node v there is at least one subgraph G_i that contains all neighbors within distance r,*
- *each node v is contained in at most* 30 *subgraphs, and*
- *each subgraph has* radius *at most $\rho = 24r - 8$*
  (a graph has radius ρ if it contains a spanning tree of depth ρ).

Furthermore, such a sparse cover can be computed in polynomial time.

Note that, since each node is in at most $O(1)$ subgraphs G_i, the total size of the cover is $O(n)$.

While for minor-free graphs the best-known construction algorithm requires polynomial time, their algorithm for planar graphs appears to actually run in time $O(n \log n)$. Let $\mathcal{SNC}(n)$ denote the time required to compute sparse neighborhood covers. We state our preprocessing bounds with respect to $\mathcal{SNC}(n)$.

2.5 Exact Distance Oracles and Cycle MSSP

We use the Cycle Multiple-Source Shortest Paths (MSSP) data structure [MS12], which is a variant of Klein's MSSP data structure [Kle05].

Lemma 5 (Cycle MSSP [MS12, Theorem 4])**.** *Given a directed planar graph G on n nodes and a simple cycle C with $c = O(\sqrt{n})$ nodes, there is an algorithm that preprocesses G in $O(n \log^3 n)$ time to produce a data structure of size $O(n \log \log c)$ that can answer the following queries in $O(c \log^2 c \log \log c)$ time: for a query node u, output the distance from u to all the nodes of C.*

We also use an exact distance oracle for small subgraphs.

Lemma 6 ([MS12, Theorem 3])**.** *For any directed planar graph G with non-negative arc lengths, there is a data structure that supports exact distance queries in G with the following properties: the preprocessing time is $O(n \log(n) \log \log(n))$, the space required is $O(n \log \log n)$, and the query time is $O(\sqrt{n} \log^2(n) \log \log(n))$.*

3 A More Compact Distance Oracle with Constant Stretch

The overall structure of the distance oracle presented in this section is also reused for the $(1+\epsilon)$–approximate distance oracle. The basic idea is to use Thorup's distance labels for long-range distances, which are the distances of length at least roughly $\log n$, and to use sparse neighborhood covers (Section 2.4) for $\log \log n$ different levels to approximate short-range distances.

We prove Proposition 1, which we restate here.

Proposition 1. *For any undirected planar graph G on n nodes with average weight bounded by $\log^\theta n$ for some constant $\theta \geq 0$ there exists an $O(1)$–approximate distance oracle with query time $O(\log\log\log n)$ using space $O(n \log\log n)$, and preprocessing time $O(n\log^3 n + \mathcal{SNC}(n)\cdot\log\log n)$, where $\mathcal{SNC}(n)$ denotes the time required to compute a sparse neighborhood cover.*

3.1 Preprocessing Algorithm

Let $\epsilon = 0.5$ (any constant $\epsilon \in (0, 1/2)$ works). The algorithm is described by the following pseudocode.

preprocess $G = (V, E)$
(i) Preparing for Long-range Queries
compute a δ–dominating set L with $\delta = \lfloor \epsilon^{-1} \log n \rfloor$ as in [KP98]
    (for graphs with edge weights s.t. $\sum_{e\in E} w(e) \leq O(n\log^\theta n)$, set $\delta = \lfloor \epsilon^{-1}\log^{\theta+1} n\rfloor$ instead
    and replace each edge e by $w(e)$ edges before computing the δ–dominating set)
for each node $l \in L$
    compute Thorup's distance label [Tho04] (see Lemma 2)
for each node $v \in V$
    compute its nearest *landmark* node l_v (the node $l_v \in L$ that minimizes $d_G(v, l_v)$)
    store $(l_v, d_G(v, l_v))$
(ii) Preparing for Short-range Queries
for every integer $i > 0$ with $2^i \leq \lceil 2\epsilon^{-1}\delta \rceil$
    compute a sparse neighborhood cover with radius $r = 2^i$ as in [BLT07]
    let $\mathcal{G}^i = \{G^i_j\}$ denote this cover
    for each node $v \in V$
        store the list of graphs $G^i_j \in \mathcal{G}^i$ with $v \in V(G^i_j)$

Space requirements Each distance label has size $O(\epsilon^{-1}\log n)$ (see Lemma 2). The space requirement for the data structure computed in the first step is thus $O(n)$. In the second step, we iterate through $O(\log(\epsilon^{-2}\log n)) = O(\log\log(n) + \log(1/\epsilon))$ levels (for weighted graphs with $\sum w(e) \leq O(n\log^\theta n)$ for a constant $\theta \geq 0$ we have $O(\theta\log\log n + \log(1/\epsilon))$ levels). At each level i, for each node, we store a list of graphs $L^i(v) \subseteq \mathcal{G}^i$ of constant length $\left|L^i(v)\right| = O(1)$ [BLT07]. Over all levels, the space requirement is thus $O(n\log(\epsilon^{-2}\log n))$. Here we assume that identifiers of length $O(\log n)$ bits can be stored using constant space, which is a common assumption in the *word RAM model* [CR73].

Preprocessing time The preprocessing time is dominated by the time required to compute the neighborhood covers $\mathcal{SNC}(n)$, which is bounded by a polynomial in the number of nodes n [BLT07]. The dominating set [KP98], the distance labels [Tho04], and the nearest landmarks [Erw00] can be computed in almost linear time in n.

3.2 Query Algorithm

The algorithm is described by the following pseudocode.

query (u, v)
return the minimum of the long-range and the short-range query algorithm
(i) Long-range Query
    return $d_G(u, l_u) + \tilde{d}_G(l_u, l_v) + d_G(l_v, v)$, where $\tilde{d}_G(\cdot,\cdot)$ is the estimate obtained from the labels
(ii) Short-range Query
binary search for a level i such that $\exists G^i_j \in \mathcal{G}^i : u, v \in V(G^i_j)$ *and* $\nexists G^{i-1}_{j'} \in \mathcal{G}^{i-1} : u, v \in V(G^{i-1}_{j'})$
    return $2\rho 2^i$, where ρ is the constant for the *radius* in Lemma 4 (here: $\rho = 24$)

Running time Computing the long-range result requires time $O(1/\epsilon)$ [Tho04]. For the short-range pairs, a binary search among $O(\log(\epsilon^{-2}\log n))$ levels can be done in time $O(\log\log(\epsilon^{-2}\log n))$. At each search level we need to compute the intersection of two sets of constant size (recall that each node is in at most $O(1)$ graphs G^i_j per level i [BLT07]).

Stretch analysis For any pair of nodes (u,v) at distance $d_G(u,v) \geq \epsilon^{-1}\delta = \epsilon^{-2}\log n$, the long-range algorithm returns a $(1+6\epsilon)$–approximation for $d_G(u,v)$, since, using the triangle inequality,

$$\begin{aligned} d_G(u,v) &\leq d_G(u,l_u) + \tilde{d}_G(l_u,l_v) + d_G(l_v,v) \\ &\leq \delta + (1+\epsilon)d_G(l_u,l_v) + \delta \\ &\leq \delta + (1+\epsilon)(\delta + d_G(u,v) + \delta) + \delta \\ &\leq (1+\epsilon)d_G(u,v) + (4+2\epsilon)\delta. \end{aligned}$$

For nodes at distance $d_G(u,v) < \epsilon^{-1}\delta$, the short-range algorithm returns a 4ρ–approximation (recall that ρ is the constant for the *radius* in [BLT07]). The graph G^i_j with $u,v \in V(G^i_j)$ at level i is a certificate that $d_G(u,v) \leq 2\rho 2^i$. Since there is no graph $G^{i-1}_{j'}$ with $u,v \in V(G^{i-1}_{j'})$ at level $i-1$, the u-to-v distance satisfies $d_G(u,v) > 2^{i-1}$.

We conclude this section by noting that, using Lemma 4, we have $\rho = 24$ and thus the stretch of this oracle is at most 96. The construction of Busch et al. [BLT07] works for general minor-free graphs. There may be a construction with smaller radius for planar graphs.

4 $(1+\epsilon)$–Approximate Distance Oracle

We prove the main result (Theorem 2). The distance oracle presented in this section is based on the oracle with constant stretch as described in Section 3.

4.1 Overview

We first run the preprocessing algorithm of Section 3 with ϵ being the actual value chosen by the application divided by 6. Note that the only distances approximated with stretch more than $1+\epsilon$ are the ones in the range $[1, \epsilon^{-2}\log(n)]$. This range of logarithmic size is then handled by data structures for log-logarithmically many levels. At level i, we are interested in distances in the range $[2^i, 2^{i+1})$.

The constant-stretch distance oracle uses a very crude estimate: if two nodes are contained in the same graph G^i_j with diameter $O(2^i)$ but they are not together in a graph at level $i-1$, they must be at distance $\Theta(2^i)$ ($O(2^i)$ due to the diameter of G^i_j and $\Omega(2^i)$ since the two nodes were not together in any graph at level $i-1$). In the following, we provide an oracle that outputs a more precise estimate for pairs of nodes at distance $\Theta(2^i)$.

We are aware of exact oracles for planar graphs that can answer *bounded-length* distance queries, which are distance queries for pairs at constant distance. For planar graphs, Kowalik and Kurowski [KK06] provide such a distance oracle that uses linear space (see [DKT10] for an extension to sparse graphs). However, we cannot use their data structures, since the query time of their oracle is exponential in the length. In our oracle, short distances may be up to *logarithmic* in n. Another approach would be to use a distance oracle for planar graphs with bounded tree-width (see [MS12]): since diameter $\Theta(2^i)$ implies tree-width $w = \Theta(2^i)$ [Epp00, DH04], the query time can be made almost proportional to w. However, here we aim at query time almost proportional to $1/\epsilon$ instead.

For the $(1+\epsilon)$–stretch oracle, we introduce an additional data structure for the level graphs G^i_j. The difference to the oracle in Section 3 — one of our main technical contributions — is the following data structure that can answer approximate distance queries with *additive stretch* $\epsilon\Delta$ for planar graphs with diameter $O(\Delta)$.

Theorem 3 (Additive-Stretch Approximate Distance Oracle)**.** *For any integer Δ, for any $\epsilon > 0$, and for any planar graph on n nodes with diameter $C \cdot \Delta$ for any constant $C > 1$ there is an approximate distance oracle with* additive stretch $\epsilon\Delta$ *using space $O(Cn \log\log(1/\epsilon) \log^*(n))$ and query time $O(C\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n))$. Furthermore, this distance oracle can be computed in time $O(Cn\epsilon^{-2} \log^3(n) \log^*(n))$.*

Let us emphasize that the oracle in Theorem 3 works for *edge-weighted* planar graphs as well. In the case of weighted graphs, we require that the *weighted* diameter (defined by the *length* of the longest shortest path) is bounded by $C \cdot \Delta$.

4.2 High-level Construction: Preprocessing and Query Algorithms

We now use the additive-stretch oracle as in Theorem 3 to prove Theorem 2, which we restate here.

Theorem 2. *For any undirected planar graph G on n nodes with average weight $\leq \log^\theta n$ for some constant $\theta \geq 0$, and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n) + \log\log\log(n))$ using space $O(n\,[\log\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$, and preprocessing time $O(n\epsilon^{-2} \log^3(n) \log\log(n) \log^*(n) + \mathcal{SNC}(n) \cdot \log\log n)$, where $\mathcal{SNC}(n)$ denotes the time required to compute a sparse neighborhood cover.*

Proof of Theorem 2. We describe the peprocessing and query algorithms. Since these algorithms are very similar to the algorithms in Section 3, we mainly highlight the differences.

Preprocessing Algorithm We run the preprocessing algorithm of Section 3 with ϵ being the actual value chosen by the application divided by a small constant (six suffices). For each level 2^i, for each graph G^i_j, we compute the additive-stretch oracle as in Theorem 3. The total space requirement per level is $O(n\,[\log\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$. The preprocessing time again depends on the time $\mathcal{SNC}(n)$ required to compute the sparse neighborhood covers, since, for each level 2^i, we compute a sparse neighborhood cover and the additive-stretch oracle in each subgraph (Theorem 3).

Query Algorithm We use the query algorithm of Section 3 with the following adaptation. After the binary search for the lowest level 2^i with a graph G^i_j that contains both u and v, we compute the result of the additive-stretch oracle for this level *and also* for the $\lceil \log_2(2\rho) \rceil$ levels above (ρ again denotes the radius in Lemma 4). The reason for this is that u and v may be in G^i_j at level 2^i but u and v may be at distance cD for some $c \in [1, 2\rho]$. At a lower level, it is not guaranteed that G^i_j actually contains an approximate shortest path. Since $\rho = O(1)$, the query time is at most $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n) + \log\log\log(n))$. □

4.3 Polynomial Weights

The proof of Theorem 1 is based on the proof of Theorem 2.

Proof of Theorem 1. There are two differences to the proof of Theorem 2.

Preprocessing Algorithm We need to consider $O(\log n)$ levels instead of $O(\log\log n)$ levels, since a node may be at distance $O(poly(n))$ from its nearest dominating node. The space complexity increases from $O(n\,[\log\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$ to $O(n\,[\log(n) + \log(1/\epsilon)] \log^*(n) \log\log(1/\epsilon))$. We also preprocess Thorup's distance oracle for $\epsilon = 1/2$ (any constant works). The space required for this is $O(n \log n)$ and thus the overall space does not increase any further.

Query Algorithm Instead of running a binary search to find the right level, we can now just query Thorup's distance oracle for constant ϵ to identify the right level. Everything else remains the same. The query time is thus $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n))$. □

5 Additive-Stretch Approximate Distance Oracle

This section is devoted to the proof of Theorem 3. While there are distance oracles for special cases of planar graphs and special kinds of queries [DPZ95, DPZ00, DPZ91, CX00, KK06, MS12], we unfortunately cannot use them for our purpose.

Let us first restate Theorem 3.

Theorem 3. *For any integer Δ, for any $\epsilon > 0$, and for any planar graph on n nodes with diameter $C \cdot \Delta$ for any constant $C > 1$ there is an approximate distance oracle with* additive stretch $\epsilon\Delta$ *using space $O(Cn \log\log(1/\epsilon) \log^*(n))$ and query time $O(C\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n))$. Furthermore, this distance oracle can be computed in time $O(Cn\epsilon^{-2} \log^3(n) \log^*(n))$.*

5.1 Overview

We adapt Thorup's distance oracle as follows. During preprocessing, we recursively separate the planar graph G into at least two pieces each with at most half the weight (Lemma 1). Analogously to computing an r–division [Fre87], we separate G in a way such that both *(i)* the sizes of the resulting pieces are balanced and *(ii)* the boundary ∂P of each piece P consists of at most a constant number of shortest paths (see [Tho04, Section 2.5.1] for a similar construction). We stop the separation as soon as subgraphs have logarithmic size.

There are three types of shortest $s-t$ paths we need to consider.

1. Any shortest path between nodes in two different pieces must pass through a boundary path.
2. For nodes in the same piece, the shortest path may leave through one boundary path and re-enter through another boundary path.
3. For nodes in the same piece P, the shortest path may lie entirely within P.

The third type of paths is handled by recursion.

The first two types of paths are handled as follows. Instead of letting the $s-t$ path pass through any node of to boundary, that is, any node on a separator path $q \in Q$, we restrict the possible portals on the boundary to the ϵ–cover for each path Q (which is chosen as a set of equally-spaced nodes $C(Q)$ as in Lemma 3). Since $C(Q)$ is an ϵ–cover, the error we introduce is bounded by $\epsilon \cdot \Delta$.

The savings in space can be obtained since *(i)* for each node we only need the distances to the $O(\epsilon^{-1})$ nodes in the covers on the boundary as opposed to the $O(\epsilon^{-1} \log n)$ nodes on all separator paths as in [Tho04] and *(ii)* we can compute these distances at query time using the Cycle MSSP data structure (Lemma 5).

5.2 Definitions

Throughout, pieces are called $P, P', P_i, \ldots$ and paths are called $Q, Q', Q_j, \ldots$. Let ∂P denote the set of separator paths on the boundary of a piece P. By $\#\partial P$ we denote the number of shortest paths on the boundary of a piece P. For a separator path Q, let $C(Q) \subseteq Q$ denote the *ϵ–cover* of Q, which is a set of $O(1/\epsilon)$ equally-spaced nodes on Q. We call each node $c \in C(Q)$ a *portal*. For any piece P, let $\mathcal{C}(P)$ denote the set of portals on its boundary, $\mathcal{C}(P) := \bigcup_{Q \in \partial P} C(Q)$.

5.3 Preprocessing Algorithm

We first compute a decomposition we call *shortest-path s–division* (similar to an r–division), by repeatedly using shortest-path separators (as in Lemma 1). The separator paths form a *decomposition tree*. Then, we compute an ϵ–cover for each path. We further connect each portal node in an ϵ–cover to all the portal nodes in covers on higher levels of the decomposition tree. Finally, we compute distances from nodes to their portals and store them implicitly using the Cycle MSSP data structure.

Compared to [Tho04], the main technical differences are:

- selective storing of distances to portals
  we store distances to portals in ϵ–covers only for a restricted set of nodes

- global ϵ–covers
  we compute one cover per path, as opposed to one cover per pair of path and node
  (to improve query time and space requirements)

Our preprocessing algorithm is outlined in the following pseudocode.

preprocess $H = (V, E)$
let $n = |V|$
let the set of pieces $\mathcal{P} = \{V\}$
let the *tree* of boundary paths $\mathcal{S} = \{\}$

($\star$) beginning of partitioning (*shortest-path division*, similar to an r–division)
WHILE there is a piece $P \in \mathcal{P}$ with $\#\partial P > 10$ or size $|P| > \lceil \epsilon^{-2} \log n \rceil$
    we compute a balanced separation with node weights as follows:
    IF a piece has too many boundary paths, $\#\partial P > 10$
        assign weight 1 to each endpoint of any boundary path $Q \in \partial P$
        assign weight 0 to all the remaining nodes
    ELSE
        assign weight 1 to each node
    let $T(u), T(v), T(w)$ be the three paths obtained by Lemma 1 applied to P, weighted as above
    add $T(u), T(v), T(w)$ to the set of separator paths $\mathcal{S}$
    remove P from $\mathcal{P}$, partition P into pieces $P_1, P_2, \ldots$ and add them to $\mathcal{P}$
($\star$) partitioning with $O(1)$ boundary paths per region computed

(†) inter-piece approximate distance oracle
FOR EACH separator path $Q \in \mathcal{S}$
    compute an ϵ–cover $C(Q) \subseteq Q$ (Lemma 3)
    FOR EACH portal $c \in C(Q)$
        compute and store the distances to all portals on *ancestor paths* Q'
FOR EACH piece $P \in \mathcal{P}$ and separator path $Q \in \partial P$
    augment the graph induced by the piece P with infinite edges such that $C(Q)$ form the outer face
    compute the Cycle MSSP Data Structure for $C(Q)$ and this augmented graph (as in Lemma 5)
(†) inter-piece oracle computed

(‡) recursive call
FOR EACH piece $P \in \mathcal{P}$
    IF $|P| > \lceil \epsilon^{-2} \rceil$
        recurse on the graph induced by P
    ELSE
        compute an exact distance oracle for P as in Lemma 6

Claim 1 (Shortest-path separator r–division)**.** *At step ($\star$) we have obtained a partitioning of the vertex set V into pieces $P_1, P_2, \ldots$ with the following properties:*

- *each piece P_i has size $|P_i| = O(\epsilon^{-1} \log n)$,*
- *the number of boundary paths $\#\partial P_i$ for each piece P_i is at most ten, and*
- *the total number of pieces is at most $O(n\epsilon / \log n)$.*

Proof. Our construction is similar to that of an r–division [Fre87, WN10b] but using shortest-path separators instead of cycle separators. It is also essentially the same as in [Tho04, Section 2.5.1]; see [Tho04, Fig. 4] for an illustration. Similar divisions have been used in [AGK+98, GKP95]. We recursively separate V into pieces using root-paths of a single shortest-path tree as in Lemma 1.

Whenever a the boundary of a piece consists of more than 10 root-paths, we separate it using Lemma 1 such that the boundary paths will be partitioned in a balanced way. Since we always use the same shortest-path tree T, we can use the following vertex weights: all the endpoints (leafs in T) of a previously selected root-path are weighted with one and all the remaining nodes are weighted with zero.

The third property can be proven using the following observation. Let us track a piece P during the execution of the preprocessing algorithm. Since all the separator paths are taken from the same shortest-path tree T, the number of boundary paths $\#\partial P$ increases only if P is the reason why Lemma 1 is invoked. A piece P' resulting from that invocation can have more than 10 boundary nodes by inheriting $10-1$ paths from P and gaining 3 paths from Lemma 1. By one more invocation of Lemma 1, the number of boundary paths of P' can be reduced to less than 10. When the recursion stops at P', either P' or P has size $\Theta(\epsilon^{-1}\log n)$. □

Claim 2. *The total space requirement per recursion level is at most $O(n\log\log(1/\epsilon))$.*

Proof. At each recursion level, for each portal in an ϵ–cover $c \in Q$, the above algorithm stores $O(\epsilon^{-1}\log n)$ portals and the corresponding distances, since, for each portal, we store the distances to all the portals on higher levels (as in [Tho04]).

By Claim 1, the total number of pieces per level is at most $O(n\epsilon^2/\log n)$ (*sic!*). Each piece is surrounded by at most 10 paths on each of which we have $O(1/\epsilon)$ portals. The total number of portals per level is thus at most $O(n\epsilon/\log n)$. Since each distance label requires space $O(\epsilon^{-1}\log n)$, the total space per level for this step is $O(n)$.

At each recursion level, for each node $v \in V$, the above algorithm stores a Cycle MSSP data structure for the portals on each boundary path (note that we can only do so since there is *one fixed* cover per boundary path as opposed to one cover per pair of node and boundary path). Since the cycle has $O(1/\epsilon)$ nodes, one Cycle MSSP data structure requires space $O(n\log\log(1/\epsilon))$ (see Lemma 5). By Claim 1, each piece has at most ten boundary paths.

On the lowest level, we also store a distance oracle for a planar graph on $O(\epsilon^{-2})$ nodes. The overall space requirement for all these distance oracles is $O(n\log\log(1/\epsilon))$ (Lemma 6). □

Note that this recursive algorithm reduces the graph size from $n \rightsquigarrow \log n$ at each level. It follows directly that the recursion depth is at most $O(\log^* n)$. The data structure computed by the above algorithm thus occupies space $O(n\log\log(1/\epsilon)\log^* n)$.

5.4 Query Algorithm

The query algorithm, given a pair of nodes u, v, returns an estimate for $d_G(u,v)$. The main differences to Thorup's distance oracle [Tho04] are:

- two-way approximation
  instead of approximating the distance by $d_G(u,q) + d_Q(q,q') + d_G(q',v)$ for two portals q, q' on a separator path as in Thorup's distance oracle, we use the estimate $d_G(u,c_u) + d_G(c_u,q)$ for a portal c_u to approximate $d_G(u,q)$ and, analogously, for $d_G(q,v)$ (by doing so, each node only needs to compute and encode distances to portals on its boundary paths as opposed to *all* $\log n$ levels)

- Monge search for the optimal portal
  since we have only *one* ϵ–cover per path, we can use the non-crossing property as in [AKM+87, KK90, FR06] to compute the two-way approximation
  we emphasize that there is also only one ϵ–cover per path on higher levels (not just at the boundary of a piece)

Let us momentarily assume that the two query nodes u and v are in different pieces P_u and P_v, respectively.[3] Since we store distances from u to its cover $\mathcal{C}(P_u)$ (and from v to $\mathcal{C}(P_v)$) during preprocessing, we may return the minimum

$$\min_{c_u \in \mathcal{C}(P_u), c_v \in \mathcal{C}(P_v)} d_G(u, c_u) + \tilde{d}(c_u, c_v) + d_G(c_v, v),$$

where $\tilde{d}(\cdot, \cdot)$ is a $(1+\epsilon)$–approximation for $d_G(\cdot, \cdot)$. We can compute $\tilde{d}(c_u, c_v)$ for the two portals c_u, c_v since they have a lowest common ancestor in the separator decomposition tree consisting of at most three shortest paths, to whose ϵ–covers we computed and stored the shortest-path distances during preprocessing.

Claim 3. *The above estimate is at most $6\epsilon\Delta$ longer than $d_G(u, v)$.*

Proof. Any shortest path from u to v must intersect the boundary of both pieces. Our approximation uses at most one additional separator path (one of the paths in the least common ancestor separator). Due to Lemma 3, the additive distortion is at most $2\epsilon\Delta$ per path. □

Per level i, we compute the minimum $\min_{c_u \in \mathcal{C}(P_u), c_v \in \mathcal{C}(P_v)} d_G(u, c_u) + \tilde{d}(c_u, c_v) + d_G(c_v, v)$. We know the distance from u to the $O(1/\epsilon)$ portals $c_u \in \mathcal{C}(P_u)$ on its boundary paths. We also know the distance from these portals to the portals on the separator paths. We wish to efficiently compute the distance from u to the portals on the relevant separator paths to simultaneously compute *all* the *relevant* $\tilde{d}(c_u, c_v)$. This computation can be done using the efficient Dijkstra implementation of [FR06].

Fakcharoenphol and Rao [FR06, Section 4.1 (and also Sec. 2.3)] devised an ingenious implementation of Dijkstra's algorithm [Dij59] that computes a shortest-path tree of a complete bipartite graph $H_1 \cup H_2$ in time $O(h \log h)$ where $h = |H_1| + |H_2|$ (as opposed to $O(|H_1| \cdot |H_2|)$) as long as the edge weights obey the Monge property (which essentially means that they correspond to distances in a planar graph wherein the nodes of H_1 and H_2 lie on a constant number of faces). We refer to this implementation as *FR-Dijkstra*.

The query algorithm works as described in the following pseudocode.

query(u, v)
return the minimum among the results from all recursion levels:
For Each recursion level
  let $u \in P_u$ and $v \in P_v$
  For Each pair of boundary paths $(Q_u, Q_v) \in \partial P_u \times \partial P_v$
    determine the three separator paths Q_1, Q_2, Q_3 that separate Q_u from Q_v (as in [Tho04])
    For Each $Q \in \{Q_1, Q_2, Q_3\}$
      compute $\min_{c_u \in C(Q_u)} d(u, q)$ for all $q \in Q$ simultaneously using FR-Dijkstra
      (analogously for v)
      compute $\min_{q \in Q} d(u, q) + d(q, v)$ and keep it if it is the new minimum
at the lowest recursion level
  If $P_u = P_v$ Then
    query the exact distance oracle

Claim 4. *The query algorithm runs in time $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n))$.*

Proof. We compute the distances from u to the portals on the boundary path $C(Q_u)$ using the Cycle MSSP data structure in time $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon))$ (Lemma 5).

For each pair of boundary paths $(Q_u, Q_v) \in \partial P_u \times \partial P_v$ there is a constant number of separator paths Q on higher levels we need to consider (see constant query time oracle in [Tho04]).

[3] Even if the two query nodes are in the same piece, it could still be that the shortest path passes through the separator (which consists of at most ten shortest paths). To account for these cases, we compute the minimum among paths passing through the separator (this computation is the same as if the nodes were in different pieces). Then we recurse until some termination criterion on the size is met.

Per separator path Q the FR-Dijkstra algorithm runs in time $O((|C(Q_u)| + |C(Q)|)\log(|C(Q_u)| + |C(Q)|)) = O(\epsilon^{-1}\log(1/\epsilon))$. (Since we do not actually need the on-line version of the bipartite Monge search in [FR06], the $\log(1/\epsilon)$ factor can be eliminated by using [AKM+87, KK90] instead. However, we use FR-Dijkstra within the Cycle MSSP data structure, whose query time currently dominates the overall query time anyway.)

This search is done for all the $O(\log^* n)$ levels of the recursion. On the lowest level, we also query the exact distance oracle in time $O(\epsilon^{-1}\log^2(1/\epsilon)\log\log(1/\epsilon))$ (Lemma 6). □

6 Conclusion

Our $(1+\epsilon)$–approximate distance oracle for planar graphs has a better space-time tradeoff — both in terms of n and ϵ — than previous oracles. The improved tradeoff currently comes at a cost: the oracle cannot be distributed as a labeling scheme (it is however not clear whether $o(\log^2 n)$ bit approximate distance labels for planar graphs exist at all) and, for the improvement in terms of n, there is a dependency on the largest edge weights (Thorup's oracle depends on the largest integer weight for digraphs only). We believe that these sacrifices may be worthwhile since they allow us to achieve almost linear dependency on $1/\epsilon$ and remove the notorious logarithmic dependency on n, contributing to an important next step towards a linear-space approximate distance oracle with almost constant query time.

An intermediate goal could be to find a linear-space distance oracle with arbitrary constant stretch and (almost) constant query time. For our construction, the overhead required to transform a constant-stretch oracle into a $(1+\epsilon)$–stretch oracle is proportional to only $\log^*(n)$ (and some overhead in $\log(1/\epsilon)$). Although our construction of the $(1+\epsilon)$–stretch oracle depends on the specific features of the constant-stretch oracle, given a better constant-stretch approximate distance oracle, it may be quite possible to improve our $(1+\epsilon)$–stretch oracle as well.

Acknowledgments

Many thanks to Mikkel Thorup for encouraging our research on these tradeoffs and for the valuable comments on an earlier version of this work.

References

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, 1998. Announced at FOCS 1993.

[ACC+96] Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, pages 514–528, 1996.

[AGK+98] Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *SODA*, pages 33–41, 1998.

[AGMW07] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. In *SPAA 2007: Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, California, USA, June 9-11, 2007*, pages 16–24, 2007.

[AKM+87] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. Announced at SoCG 1986.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, 22-24 October 1990, St. Louis, Missouri, USA*, pages 503–513, 1990.

[BBJ+02] Christopher L. Barrett, Keith R. 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 *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, pages 126–138, 2002.

[BG07] Zachary K. Baker and Maya Gokhale. On the acceleration of shortest path calculations in transportatoin networks. In *International Symposium on Field-Programmable Custom Computing Machines*, pages 23–32, 2007.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 61–70, 2007.

[Cab06] Sergio Cabello. Many distances in planar graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1213–1220, 2006. A preprint of the journal version is available in the University of Ljubljana preprint series, Vol. 47 (2009), 1089.

[CCRK04] Manuel Costa, Miguel Castro, Antony I. T. Rowstron, and Peter B. Key. Pic: Practical internet coordinates for distance estimation. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan*, pages 178–187, 2004.

[Che95] Danny Ziyi Chen. On the all-pairs euclidean short path problem. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 292–301, 1995.

[CR73] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973. Announced at STOC 1972.

[CX00] Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

[CZE+11] Leonid Chindelevitch, Daniel Ziemek, Ahmed Enayetallah, Ranjit Randhawa, Ben Sidders, Christoph Brockel, and Enoch Huang. Causal reasoning on biological networks: Interpreting transcriptional changes. In *Research in Computational Molecular Biology, 15th Annual International Conference, RECOMB 2011*, 2011.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.

[DF85] Martin E. Dyer and Alan M. Frieze. A simple heuristic for the p–centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[DGJ08] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*. 2008.

[DH04] Erik D. Demaine and Mohammad Taghi Hajiaghayi. Diameter and treewidth in minor-closed graph families, revisited. *Algorithmica*, 40(3):211–215, 2004.

[Die05] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[Dij59] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Dji96] Hristo Nikolov Djidjev. Efficient algorithms for shortest path problems on planar digraphs. In *Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96, Cadenabbia (Como), Italy, June 12-14, 1996, Proceedings*, pages 151–165, 1996.

[DKT10] Zdenek Dvorak, Daniel Král, and Robin Thomas. Deciding first-order properties for sparse graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 133–142, 2010.

[DPZ91] Hristo Nikolov Djidjev, Grammati E. Pantziou, and Christos D. Zaroliagis. Computing shortest paths and distances in planar graphs. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, pages 327–338, 1991.

[DPZ95] Hristo Nikolov Djidjev, Grammati E. Pantziou, and Christos D. Zaroliagis. On-line and dynamic algorithms for shorted path problems. In *STACS*, pages 193–204, 1995.

[DPZ00] Hristo Nikolov Djidjev, Grammati E. Pantziou, and Christos D. Zaroliagis. Improved algorithms for dynamic shortest paths. *Algorithmica*, 28(4):367–389, 2000.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks - Design, Analysis, and Simulation [DFG priority program 1126]*, pages 117–139, 2009.

[EBN09] Brian Eriksson, Paul Barford, and Robert D. Nowak. Estimating hop distance between arbitrary host pairs. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 801–809, 2009.

[Epp00] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, 2000.

[Erw00] Martin Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[FJ89] Greg N. Frederickson and Ravi Janardan. Efficient message routing in planar networks. *SIAM J. Comput.*, 18(4):843–857, 1989.

[FJ90] Greg N. Frederickson and Ravi Janardan. Space-efficient message routing in c-decomposable networks. *SIAM J. Comput.*, 19(1):164–181, 1990.

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Announced at FOCS 2001.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[GKK+01] Cyril Gavoille, Michal Katz, Nir A. Katz, Christophe Paul, and David Peleg. Approximate distance labeling schemes. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 476–487, 2001.

[GKP95] Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph tsp. In *FOCS*, pages 640–645, 1995.

[GKR01] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). In *FOCS*, pages 148–157, 2001.

[Gol07] Andrew V. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, pages 88–102, 2007.

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004. Announced at SODA 2001.

[HKRS97] Monika Rauch Henzinger, Philip Nathan Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. Announced at STOC 1994.

[JHR96] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12 - 16, 1996, Rockville, Maryland, USA*, pages 261–268, 1996.

[Kar29] Frigyes Karinthy. *Lancszemek*. 1929.

[KK90] Maria M. Klawe and Daniel J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM J. Discrete Math.*, 3(1):81–97, 1990.

[KK06] Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms*, 2(3):335–363, 2006. Announced at STOC 2003.

[KKS11] Kenichi Kawarabayashi, Philip Nathan Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *Automata, Languages and Programming, 38th International Colloquium (ICALP)*, pages 135–146, 2011.

[Kle02] Philip Nathan Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 820–827, 2002.

[Kle05] Philip Nathan Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005.

[KLMN04] Robert Krauthgamer, James R. Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 434–443, 2004.

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998. Announced at PODC 1995.

[KS98] Philip Nathan Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Mil67] Stanley Milgram. The small world problem. *Psychology Today*, 1:61–67, 1967.

[Mil86] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. Announced at STOC 1984.

[MS12] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, 2012. to appear, preprint available on the arXiv http://arxiv.org/abs/1011.5549.

[New01] Mark E. J. Newman. Scientific collaboration networks. II. shortest paths, weighted networks, and centrality. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 64, 2001.

[Nus11] Yahav Nussbaum. Improved distance queries in planar graphs. In *12th International Symposium on Algorithms and Data Structures (WADS)*, pages 642–653, 2011.

[NZ02] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, 2002.

[Pel00] David Peleg. Proximity-preserving labeling schemes. *J. Graph Theory*, 33(3):167–176, 2000. Announced at WG 1999.

[Ple80] Ján Plesník. On the computational complexity of centers locating in a graph. *Applications of Mathematics*, 25(6):445–452, 1980.

[Ple87] Ján Plesník. A heuristic for the p–center problems in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987.

[PR10] Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup–Zwick bound. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, 2010.

[RN04] Bryan Raney and Kai Nagel. Iterative route planning for large-scale modular transportation simulations. *Future Generation Computer Systems*, 20(7):1101–1118, 2004.

[SS80] Mischa Schwartz and Thomas E. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications*, 28(4):539–552, Apr 1980.

[ST08] Yuval Shavitt and Tomer Tankel. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Trans. Netw.*, 16:25–36, February 2008.

[Sto99] Bryan Stout. Smart move: Intelligent path-finding. Online at gamasutra.com/view/feature/3317/smart_move_intelligent_.php, 1999.

[STW04] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. HOPI: An efficient connection index for complex XML document collections. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*, pages 237–255, 2004.

[STW05] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 360–371, 2005.

[SVY09] Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 2009.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004. Announced at FOCS 2001.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005. Announced at STOC 2001.

[Ung51] Peter Ungar. A theorem on planar graphs. *Journal of the London Mathematical Society*, s1-26(4):256–262, 1951.

[WN10a] Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time, 2010. Computational Geometry, special issue on the 25th European Workshop on Computational Geometry (to appear).

[WN10b] Christian Wulff-Nilsen. Min st-cut of a planar graph in $O(n \log \log n)$ time. *CoRR*, abs/1007.3609, 2010.

[Zar08] Christos Zaroliagis. Engineering algorithms for large network applications. In *Encyclopedia of Algorithms*. 2008.

[ZKM97] Athanasios K. Ziliaskopoulos, Dimitri Kotzinos, and Hani S. Mahmassani. Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications. *Transportation Research Part C: Emerging Technologies*, 5(2):95–107, 1997.