More Compact Oracles for Approximate Distances in Undirected Planar Graphs

Ken-ichi Kawarabayashi* k_keniti@nii.ac.jp
Christian Sommer csom@csail.mit.edu
Mikkel Thorup† mthorup@diku.dk

September 30, 2012

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. Unless stated otherwise, we assume all graphs to be planar and undirected.

In FOCS 2001 (J. ACM 2004), Thorup introduced approximate distance oracles for planar graphs (concurrent with Klein, SODA 2002). Thorup proved that, for any $\epsilon > 0$ and for any undirected planar graph $G = (V, E)$ on $n = |V|$ 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 $O(\epsilon^{-1})$.

In this paper, we aim at reducing the polynomial dependency on ϵ^{-1} and $\log n$, getting the first improvement in the query time–space tradeoff. To simplify the statement of our bounds, we define $\bar{O}(\cdot)$ to hide $\log \log n$ and $\log(1/\epsilon)$ factors.

- We provide the first oracle with a time–space product that is subquadratic in ϵ^{-1}. We obtain an oracle with space $\bar{O}(n \log n)$ and query time $\bar{O}(\epsilon^{-1})$.
- For unweighted graphs we show how the logarithmic dependency on n can be removed. We obtain an oracle with space $\bar{O}(n)$ and query time $\bar{O}(\epsilon^{-1})$. This bound also holds for graphs with polylogarithmic average edge length, which may be a quite reasonable assumption, e.g., for road networks.

1 Introduction

Distance oracles [TZ05] generalize the all-pairs shortest paths problem as follows: instead of computing and storing a distance matrix (using quadratic space and pairwise distance computations with *one* table look-up), we wish to compute a data structure that requires sub-quadratic *space* S but still allows for *efficient* (as in sub-linear *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 route planning and navigation [Gol07, Zar08, DSSW09], Geographic Information Systems (GIS) and intelligent transportation systems [JHR96], logistics, traffic simulations [ZKM97, RN04, BG07], social networks [New01, DGNP10], and others.

Distance oracles were introduced in 2001 both for general graphs [TZ05] and for planar graphs (Thorup [Tho04] and Klein [Kle02]). For general graphs, there have been many extensions and improvements [RTZ05, MN07, MS09, BK06, BS06, BGSU08, PR10, AG11, PR11, WN12a, PRT12, WN12b], see e.g. [Som10, Som12] for an overview. Yet, these general 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 better tradeoffs is that algorithms can make use of small *separators* [Ung51, LT79, Mil86]. For approximate distance oracles, separators consisting itself of few shortest paths are particularly useful [Tho04, Kle02]. For planar graphs, the best *exact* distance oracles [FR06, Dji96, CX00, Cab12, Nus11, MS12]

*NII and JST, ERATO Kawarabayashi Project
†AT&T and University of Copenhagen

Space	Query	α	dir	$\ell(e)$	Reference
$O(n)$	$O(n)$	1	$\rightarrow$	any	SSSP [HKRS97]
$O(n^2)$	$O(1)$	1	$\rightarrow$	any	APSP
$o(n^2)$	$O(1)$	1	—	$O(1)$	[WN10b] (extensions in [WN10a])
$O(S)$	$\widetilde{O}(n/\sqrt{S})$	1	$\rightarrow$	any	[MS12] for any $S \in [n \log\log n, n^2]$
$O(n \log n)$	$\widetilde{O}(\sqrt{n})$	1	$\rightarrow$	any	[FR06]
$O(n)$	$O(n^{(1/2)+\delta})$	1	$\rightarrow$	any	[MS12] for any constant $\delta > 0$
$O(n\epsilon^{-1} \log^2 n)$	$\bar{O}(\epsilon^{-1})$	$1+\epsilon$	$\rightarrow$	$n^{O(1)}$	[Tho04]
$O(n\epsilon^{-1} \log n)$	$O(\epsilon^{-1})$	$1+\epsilon$	—	any	[Tho04]
$O(n)$	$O(\epsilon^{-2} \log^2 n)$	$1+\epsilon$	—	any	[KKS11]
$\bar{O}(n \log n)$	$\bar{O}(\epsilon^{-1})$	$1+\epsilon$	—	any	Theorem 1.1
$\bar{O}(n)$	$\bar{O}(\epsilon^{-1})$	$1+\epsilon$	—	$\log^{O(1)} n$	Theorem 1.2

Table 1: Space and time complexities of distance oracles for undirected (—) planar graphs (some results extend to planar digraphs ($\rightarrow$)). $\bar{O}(\cdot)$ hides $\log\log n$ and $\log(1/\epsilon)$ factors. As usual, $\widetilde{O}(\cdot)$ hides $\log n$ factors.

have a space–query time product $S \cdot Q$ proportional to roughly $n\sqrt{n}$ (similar practical methods have been proposed [SWZ02, HSW08, DHM+09, DGPW11]). The best *approximate* distance oracle is by Thorup [Tho04] (experimental results in [MZ07]), where the product $S \cdot Q$ is $O(n\epsilon^{-2} \log n)$. Except for preprocessing [Kle05], no improvements have been made in the last decade.[1] Since separator-based approaches often use recursion of logarithmic depth, which manifests itself by logarithmic factors in either the space requirement or the query time (or even both), these results may have been perceived as optimal (at least in terms of n).

Contributions. Our distance oracles provide the first improvements upon Thorup's tradeoffs. For undirected graphs, we obtain improvements both in terms of n and ϵ, getting close to linear space, almost without affecting query times. It is possible to reduce the space, however, until the current work, only at the cost of increasing the query time [KKS11]. For an overview and comparison of existing and new results, see Table 1.

We provide the first oracle with a space–query time product that has a *subquadratic* dependency on $1/\epsilon$ (known constructions have ϵ^{-1} in both space and query [Tho04, Kle02] or ϵ^{-2} in the query complexity [KKS11]). We essentially show how to eliminate the ϵ^{-1} term in the space complexity of Thorup's oracle — almost without increasing the query time.

THEOREM 1.1. *For any undirected planar graph G on n nodes with edge lengths polynomial in n and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle with query time $\bar{O}(\epsilon^{-1})$, using space $\bar{O}(n \log n)$, and preprocessing time $\bar{O}(n\epsilon^{-2} \log^4 n)$.*

We further provide improved bounds for graphs with *moderate* edge lengths, where by moderate we mean that, after normalization such that the shortest edge length is 1, the average length is bounded by $poly(\log n)$. We believe that this is a reasonable assumption as the average length for the European road network (the version made available for scientific use by the company PTV AG) appears to be not too large.[2]

THEOREM 1.2. *For any undirected planar graph G on n nodes with average polylogarithmic edge length and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle with query time $\bar{O}(\epsilon^{-1})$, using space $\bar{O}(n)$, and preprocessing time $\bar{O}(n\epsilon^{-2} \log^3 n)$.*

Previously, decreasing the space to linear implied increasing the query time to $Q = O(\epsilon^{-2} \log^2 n)$ [KKS11].

Overall Approach. Our constructions are based on two main building blocks, both of which are oracles that occupy only close-to-linear space:

[1] Recently, Abraham, Chechik, and Gavoille [ACG12] improved upon the *dynamic* oracle of Klein and Subramanian [KS98].

[2] The network has 18,010,173 nodes and 42,560,279 edges, covering 14 European countries, serving as an important benchmark instance for shortest-path query methods [DGJ08]. For the travel-time metric (`scc-eur2time.gr`), the total length is $\sum \ell(e) = 21{,}340{,}824{,}356$, which yields an average of approximately 501. For the distance metric (`scc-eur2dist.gr`) the two values are $\sum = 9{,}420{,}195{,}951$ and $\sum / m \approx 221$. We do not claim to rigorously distinguish $O(poly(\log |E|))$ from $\Omega(\sqrt{|E|})$ for this E (here $\sqrt{42{,}560{,}279} \approx 6{,}523.82$). We however observe that $\log_2 |E| = \log_2(42{,}560{,}279) \approx 25.34$ and thus $(\log_2 |E|)^2 \geq 501$.

(i) the first distance oracle with *additive* stretch and *(ii)* an oracle with constant multiplicative stretch. Both data structures are more compact than Thorup's original data structure. The main results above are then obtained by combining these building blocks in various ways.

As a first building block, we provide a distance oracle with *additive* stretch, which is one of our main technical contributions. For a graph with diameter $O(\Delta)$, and for any $\epsilon > 0$, we can build an oracle with additive stretch $\epsilon\Delta$ using almost linear space and almost constant query time.

THEOREM 1.3. *For any integer Δ, for any $\epsilon > 0$, and for any undirected planar graph on n nodes with diameter $O(\Delta)$, there is an approximate distance oracle with* additive stretch $\epsilon\Delta$ *using space $\tilde{O}(n)$ and query time $\tilde{O}(\epsilon^{-1})$. Furthermore, this distance oracle can be computed in time $\tilde{O}(n\epsilon^{-2}\log^3 n)$.*

Let us emphasize that the oracle in Theorem 1.3 works for planar graphs with arbitrary non-negative edge lengths. In the case of graphs with such length functions, we require that the diameter defined by the *length* of the longest shortest path (as opposed to the number of edges) is bounded by $O(\Delta)$.

As a second building block, we provide a distance oracle with constant multiplicative stretch — while the stretch is larger than $(1+\epsilon)$, the oracle is more compact than Thorup's oracle.

THEOREM 1.4. *For any undirected planar graph G on n nodes with polylogarithmic average edge length there exists an $O(1)$–approximate distance oracle with query time $\tilde{O}(1)$, using space $\tilde{O}(n)$, and preprocessing time $O(n\log^3 n)$.*

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, GKR04] 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. We devise and demonstrate a planar-graphs analog of techniques for Euclidean settings [AGK+98, GKP95] and bounded-doubling-dimension graphs [BGK+11], by combining two techniques used by many algorithms for planar, bounded-genus, and minor-free graphs: *(i)* size reduction and *(ii)* diameter reduction. *(i)* Many exact algorithms use recursive separators (called *r–divisions* [Fre87]) to decompose the graph into pieces of size at most r. We introduce a new kind of r–divisions based on shortest-path separators, which may be useful to solve other problems on structured graphs as well. *(ii)* Many polynomial-time approximation schemes use techniques to reduce the diameter,[3] closely related to the *weak diameter decomposition* [KPR93] (we use the variant called *strong diameter decomposition*, or also *sparse neighborhood cover*, as in [BLT07, AGMW10], see Section 2.3). We combine both decompositions, thereby reducing piece sizes and subgraph diameters *simultaneously* — we hope that our approach has other applications.

Some of our techniques extend to graphs with bounded genus and to graphs excluding a fixed minor, for which $(1+\epsilon)$–approximate distance oracles based on shortest-path separators have been found as well [AG06, KKS11].

We note that similar bounds on space and query time are also known for graphs with bounded doubling dimension [HPM06, BGK+11] using rather different techniques. On a very abstract level, the point hierarchy and the coverage properties used by Bartal et al. [BGK+11] have some commonalities with our approach using size reduction and diameter reduction, the actual implementation is however entirely different and novel. We also note that the dependency on ϵ^{-1} is polynomial (actually almost linear) in our oracles, while it is often exponential for oracles processing bounded-doubling-dimension graphs.

2 Preliminaries

We use standard terminology from graph theory, see for example [Die05]. Graphs we consider are undirected and planar, and they have n nodes.

Let $\log^*(\cdot)$ denote the *iterated logarithm* function, 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 $\rho < 1$.

[3] Diameter reduction by deleting or contracting subsets of edges [Bak94, Kle08, DHK05, DHM10, DHK11] allows to exploit the linear relationship between diameter and tree-width [Bak94, Epp00, DH04].

Planar graphs are known to have *small* separators: for any planar graph there exists a balanced separator consisting of $O(\sqrt{n})$ nodes [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 2.1. (SP SEPARABILITY [THO04, L. 2.3]) *In linear time, given an undirected planar graph G 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 $G \setminus V(T(u) \cup T(v) \cup T(w))$ has at most half the weight of G.*

Since T is a shortest-path tree, this 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. In our data structures, we keep only a subset of the labels to reduce the space consumption.

LEMMA 2.2. ([THO04, THEOREMS 3.16 AND 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 for this algorithm is the concept of *shortest-path tree separability* as outlined in the previous section (Lemma 2.1). Three shortest paths Q_i 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 also use the following variant of Klein and Subramanian [KS98] (which they call a *crossing substitute*).

LEMMA 2.3. (CROSSING SUBSTITUTE [KS98, L. 4]) *Let Q be a path of length $O(D)$ for an integer D. Let $C(Q) \subseteq Q$ be a set of $O(1/\epsilon)$ equally spaced nodes on Q, called* the *ϵ–portal set of Q. Let u, v be any two nodes at distance $[D, 2D)$ whose shortest path intersects Q. The distance can be approximated using $c \in C(Q)$ s.t.*

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

The proof is based on the observation that any *detour* using $c \in C(Q)$ instead of $q \in Q$ causes an additive error of at most $O(\epsilon D)$. When there is no bound on the length of the path, there still exists an ϵ–portal set of size $O(1/\epsilon)$, however, the portal set is not global anymore, meaning that each node v uses a different $C(v,Q)$.

LEMMA 2.4. ([THO04, L. 3.4]) *For any node v and for any shortest path Q there exists a set of nodes $C(v,Q)$ such that, for any node $q \in Q$ there exists a portal $c \in C(v,Q)$ satisfying*

$$d(v,c) + d(c,q) \le (1+\epsilon) d(v,q).$$

As a consequence, for any pair of nodes (u,v) whose shortest path intersects Q, the u-to-v distance can be $(1+\epsilon)$–approximated by a path going through a node $c_u \in C(u,Q)$ and a node $c_v \in C(v,Q)$, that is,

$$d(u,v) \le \min_{c_u \in C(u,Q), c_v \in C(v,Q)} d(u,c_u) + d(c_u,c_v) + d(c_v,v) \le (1+\epsilon) d(u,v).$$

When storing the distance from v to all its portals $c \in C(v,Q)$, we store both the distance to $c \in C(v,Q)$ and also the position (distance from the root) of c on the path to efficiently compute $d(c_u,c_v)$ for two portals.

2.3 Strong Diameter Decomposition / Sparse Neighborhood Cover. Busch, LaFortune, and Tirthapura [BLT07] provide *sparse covers* [AP90, ABCP98] for planar graphs. A concurrent and independent construction with slightly

weaker constants for planar graphs but significantly better bounds for minor-free graphs is given by Abraham, Gavoille, Malkhi, and Wieder [AGMW10]. Sparse covers have found numerous applications in distributed computing.

LEMMA 2.5. (SPARSE COVER [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 its neighbors $N^r(v)$ within distance r,*
- *each node v is contained in at most* 30 *subgraphs, and*
- *each subgraph has* radius[4] *at most $\rho = 24r - 8$.*

Furthermore, such a sparse cover can be computed in time $O(n \log n)$.

Since each node is in $\Theta(1)$ subgraphs G_i, the total size of the cover is $\Theta(n)$. Polynomial construction time is claimed for minor-free graphs [BLT07, AGMW10]. For planar graphs, their algorithm actually runs in $O(n \log n)$, see appendix, Section A.

2.4 Distance–δ Dominating Sets. Let $\delta < n$ be an integer. A *δ–dominating set* of an unweighted, connected 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].

2.5 FR-Dijkstra. Fakcharoenphol and Rao [FR06, Section 4.1 (and also Sec. 2.3)] devised an ingenious implementation of Dijkstra's algorithm [Dij59] computing 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 lengths obey the Monge property (which more or less means that they correspond to distances in a planar graph wherein the nodes of H_1 and H_2 lie on $O(1)$ faces). We refer to this implementation as *FR-Dijkstra*. FR-Dijkstra can handle multiple bipartite graphs at once. We note that, for the particular case with just one bipartite graph, Djidjev [Dji96] already gave an $O(h \log h)$–time algorithm.

2.6 Exact Distance Oracles and Cycle MSSP. We use the Cycle Multiple-Source Shortest Paths (MSSP) data structure of Mozes and Sommer [MS12], which is based on Klein's MSSP data structure [Kle05].

LEMMA 2.6. (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. Both data structures internally use *FR-Dijkstra*.

LEMMA 2.7. ([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 Additive-Stretch Oracle (Theorem 1.3)

Overview. During preprocessing, we recursively separate the planar graph G into at least two pieces each with at most half the weight (Lemma 2.1). 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.[5] *(1)* Any shortest path between nodes in two different pieces must intersect with 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 intersect with any node of the boundary, that is, any node q on a separator path Q, we restrict the nodes on the boundary to a set of *portals*, a so-called *ϵ–portal set*

[4] A graph has radius ρ if it contains a spanning tree of depth ρ.

[5] For clarity of exposition, in this analysis, we assume that shortest paths are unique. Arguments generalize.

for each path Q (which is chosen as a set of equally-spaced nodes $C(Q)$ as in Lemma 2.3). Since $C(Q)$ is an ϵ–portal set, the error we introduce is bounded by $\epsilon \cdot \Delta$ (Q has length $O(\Delta)$).

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 portal sets on the boundary as opposed to the $O(\epsilon^{-1} \log n)$ portals on all separator paths and *(ii)* instead of storing these distances, we compute them at query time using Cycle MSSP (Lemma 2.6).

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–portal set* 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, $\mathcal{C}(P) := \bigcup_{Q \in \partial P} C(Q)$.

3.1 Preprocessing Algorithm. We first compute a decomposition we call *shortest-path r–division* (similar to an r–division), by repeatedly using shortest-path separators (as in Lemma 2.1). The separator paths form a *decomposition tree*. Our construction is similar to that of an r–division with $O(1)$ holes [Fre87, KS98, FR06, WN10c, KMS12] but using shortest-path separators instead of cycle separators. It is also essentially the same as in [Tho04, Section 2.5.1]; (similar divisions in [AGK+98, GKP95]).

shortest-path-division $H = (V, E)$
let $n = |V|$;
let the set of pieces $\mathcal{P} = \{V\}$
let the *tree* of boundary paths $\mathcal{S} = \{\}$
let T be a shortest-path tree (rooted arbitrarily)

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 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 3 paths obtained by Lemma 2.1 applied to P, weighted as above
  add $T(u), T(v), T(w)$ to set of sep. paths $\mathcal{S}$
  remove P from $\mathcal{P}$
  partition P into pieces $P_1, P_2, \ldots$
  add new pieces P_i to $\mathcal{P}$

CLAIM 1. (SHORTEST-PATH SEPARATOR r–DIVISION) *There is an algorithm that computes a partitioning of the vertex set V into pieces $P_1, P_2, \ldots$ with the following properties:* (i) *each piece P_i has size $|P_i| = O(\epsilon^{-2} \log n)$,* (ii) *the number of boundary paths $\#\partial P_i$ for each piece P_i is at most ten, and* (iii) *the total number of pieces is at most $O(n\epsilon^2/\log n)$.*

Proof. Whenever the boundary of a piece consists of more than 10 root-paths, we separate it using Lemma 2.1 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 2.1 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 2.1. By one more invocation of Lemma 2.1, 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 $\Theta(\epsilon^{-2} \log n)$. □

Then, we compute an ϵ–portal set for each path. We further connect each portal node in an ϵ–portal set to all the portal nodes in portal sets on higher levels of the decomposition tree (we essentially compute its distance label). Finally, we compute distances from nodes in a piece to their portals and we store them implicitly using the Cycle MSSP data structure.

Compared to [Tho04], the main technical differences are: *(i)* selective storing of distances to portals: we store distances to portals in ϵ–portal sets only for a restricted set of nodes, and *(ii)* global ϵ–portal sets: we compute one portal set per path, as opposed to one portal set 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 $\mathcal{S} = \{\}$
let T be a shortest-path tree (rooted arbitrarily)

$(\mathcal{P}, \mathcal{S}) \leftarrow$ shortest-path-division(H)

(†) inter-piece approximate distance oracle
For Each separator path $Q \in \mathcal{S}$
  compute an ϵ–portal set $C(Q) \subseteq Q$
    (crossing substitute, Lemma 2.3)
  For Each portal $c \in C(Q)$
    compute and store distances to all portals
    on *ancestor paths* Q' (distance label of c)
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 s.t. $C(Q)$ forms outer face
  compute Cycle MSSP Data Structure for $C(Q)$
  and this augmented graph (as in Lemma 2.6)
(†) inter-piece oracle computed

(‡) recursive call (if necessary)
For Each piece $P \in \mathcal{P}$
  If $|P| > \lceil \epsilon^{-2} \rceil$: recurse on P
  Else : compute exact oracle for P (Lem. 2.7)

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–portal set $c \in Q$, the above algorithm stores distance labels of size $O(\epsilon^{-1} \log n)$, since, for each portal, we store the distances to all the portals on higher levels.

By Claim 1, the total number of pieces per level is at most $O(n\epsilon^2/\log n)$. 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* portal set per boundary path as opposed to one portal set per pair of node and boundary path). Since the cycle has $O(1/\epsilon)$ nodes, one Cycle MSSP data structure requires space $O(|P| \log\log(1/\epsilon))$ for a piece P (see Lemma 2.6). By Claim 1, each piece has at most ten boundary paths.

On the lowest level, we also store an exact distance oracle for a planar graph on $O(\epsilon^{-2})$ nodes (Lemma 2.7). The overall space requirement for all these distance oracles is $O(n \log\log(1/\epsilon))$. □

This recursive algorithm reduces the graph size from $n \rightsquigarrow \log n$ at each level, hence the recursion depth is at most $O(\log^* n)$. The total space requirements are thus $O(n \log\log(1/\epsilon) \log^*(n))$.

3.2 Query Algorithm. The query algorithm, given a pair of nodes u, v, returns an estimate for $d_G(u, v)$. Compared to [Tho04], the main differences are *(i)* 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, 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), and *(ii)* Monge search for the optimal portal: since we have only *one* ϵ–portal set per path, we can use the non-crossing property as in [FR06] to compute the two-way approximation. Let us emphasize that there is also only one ϵ–portal set 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.[6] Since we store distances from u to its portal set $\mathcal{C}(P_u)$ (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–portal sets we computed and stored the shortest-path distances during preprocessing.

Claim 3. *The above estimate is at most* $6\epsilon\Delta$ *longer (additive) 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 2.3, 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

[6] Even if the two query nodes are in the same piece, it could still be that the shortest path intersects with the separator (which consists of at most ten shortest paths). To account for these cases, we compute the minimum among paths intersecting with the separator (this computation is the same as if the nodes were in different pieces). Then we recurse.

higher-level separator paths. We wish to efficiently compute the distance from u to the portals on the relevant separator paths — we simultaneously compute *all* the *relevant* $\tilde{d}(c_u, c_v)$ using FR-Dijkstra (Section 2.5).

The query algorithm works as described in the pseudocode below.

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 3 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 by FR-Dijkstra [FR06] (analogously for v)
      compute $\min_{q \in Q} d(u, q) + d(q, v)$, keep 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 2.6).

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.

Per separator path Q the FR-Dijkstra algorithm runs in time $O((|C(Q_u)| + |C(Q)|) \log(|C(Q_u)| + |C(Q)|)) = O(\epsilon^{-1} \log(1/\epsilon))$.

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 2.7). □

4 $(1+\epsilon)$–Stretch Oracle (Theorem 1.1)

Given the additive-stretch oracle (Theorem 1.3), the construction is rather straightforward. Let us first note that we may assume that $\epsilon > 1/n$. If $\epsilon \leq 1/n$, no preprocessing is required, since, at query time, we can just run the single-source shortest paths algorithm of Henzinger et al. [HKRS97].

Preprocessing Algorithm We use a scaling approach. We compute sparse neighborhood covers for $r \in \{1, 2, 4, \ldots 2^i, \ldots\}$ and, for each resulting graph G^i_j, we compute the additive-stretch oracle of Theorem 1.3. Before computing the oracle for a graph G^i_j, we slightly modify G^i_j by contracting all edges e of length $\ell(e) < r/n^2$. As a consequence, each edge e occurs in at most $O(\log n)$ different scales, independent of its length $\ell(e)$. The total space requirement is thus $O(n \log(n) \log\log(1/\epsilon) \log^*(n)) = \tilde{O}(n \log n)$.

To enable efficient identification of the right scale at query time, we preprocess Thorup's distance oracle for $\epsilon = 1/2$ (any constant works) on the entire graph. The space requirement for this step is $O(n \log n)$.

Query Algorithm Given a pair of nodes (u, v) we query the distance oracle for constant ϵ to identify the right scale i. At that scale $r = 2^i$, we query the additive-stretch oracles for all the graphs G^i_j that contain both nodes $u, v \in V(G^i_j)$, and we return the minimum distance plus $\lfloor r/n \rfloor$ (to account for $n-1$ edges that could have been contracted at preprocessing due to their length being less than r/n^2). Overall, the query time is $O(\epsilon^{-1} \log^2(1/\epsilon) \log\log(1/\epsilon) \log^*(n)) = \tilde{O}(\epsilon^{-1})$.

5 Constant-Stretch Oracle, Polylog Lengths (Theorem 1.4)

Towards achieving a more compact $(1+\epsilon)$–stretch distance oracle, we first provide an $O(1)$–stretch distance oracle. The overall construction is then reused to obtain the $(1+\epsilon)$–approximate distance oracle (Section 6). The basic idea is to use distance labels for long-range distances, which are the distances of length at least roughly $\log n$, and to use sparse neighborhood covers (Section 2.3) for $O(\log\log n)$ different levels to approximate short-range distances.

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 lengths s.t. $\sum_{e \in E} \ell(e) \leq O(n \log^\theta n)$, set $\delta = \lfloor \epsilon^{-1} \log^{\theta+1} n \rfloor$ instead and replace each edge e by $\ell(e)$ edges before computing the δ–dominating set)
For Each node $l \in L$
  compute distance labels [Tho04] (Lemma 2.2)
For Each node $v \in V$
  compute its nearest *landmark* node l_v (a 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 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 2.2). δ was chosen such that the space requirement for the data structure computed in the first step is $O(n)$. In the second step, we iterate through $O(\log(\epsilon^{-2} \log n)) = O(\log\log(n) + \log(1/\epsilon))$ levels (for graphs with lengths $\ell(e)$ such that $\sum \ell(e) \leq O(n \log^\theta n)$ 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* [Hag98], which models what can be implemented using programming languages such as C/C++.

Preprocessing time The preprocessing time is dominated by the time required to compute the distance labels [Tho04]. The dominating set [KP98], and the nearest landmark for each node can be computed in time almost linear in n. The neighborhood covers for $O(\log\log n)$ levels can be computed in time $O(n \log n)$ each [BLT07] (see also Section 2.3 and Section A).

Query Algorithm. The algorithm is described by the following pseudocode. For long-range distances, we use the labels; for short-range distances, we find the right level using binary search.

query (u, v)
return the minimum of the following long-range and short-range query computations
(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 2.5 (here: $\rho = 24$)

Running time Computing the long-range result requires time $O(1/\epsilon)$. 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[7] 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 $d_G(u, v) > 2^{i-1}$.

6 $(1+\epsilon)$–Stretch Oracle, Polylog Lengths (Theorem 1.2)

The construction of the distance oracle presented in this section is based on the construction in Section 5.

The constant-stretch distance oracle uses a very crude estimate for short-range query pairs: 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 use a more precise estimate for pairs of nodes at distance $\Theta(2^i)$, using the additive-stretch distance oracle[8] as in Theorem 1.3 for the level graphs G^i_j.

Preprocessing Algorithm We run the preprocessing algorithm of Section 5 with ϵ being

[7] Efficient set intersection is closely related to distance oracles and conjectured to require $\Omega(n^2)$ space [PR10]. Here, our sets have constant size, allowing for trivially efficient queries.

[8] 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 oracles 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 $w = O(2^i)$ [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.

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 1.3. The total space requirement per level is $O(n\,[\log\log(n) + \log(1/\epsilon)]\log^*(n)\log\log(1/\epsilon))$. The preprocessing time depends on the time 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 (which dominates the preprocessing time, Theorem 1.3).

Query Algorithm We use the query algorithm of Section 5 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 2.5). 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 $c2^i$ 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 $O(\epsilon^{-1}\log^2(1/\epsilon)\log\log(1/\epsilon)\log^*(n) + \log\log\log(n))$.

7 Conclusion

Our $(1+\epsilon)$–approximate distance oracle for planar graphs has a better space–query time tradeoff — both in terms of n and ϵ — than previous oracles. In previous work [KKS11], reducing the space to linear caused the query time to increase, in this work it does not.

The improved tradeoff currently comes at a cost: the oracle cannot be distributed as a labeling scheme. However, it seems reasonable that such a shortcoming may actually be necessary: it is questionable whether $o(\log^2 n)$–bits approximate distance labels for planar graphs exist at all. For s–approximate distance labels (constant $s > 1$) of graphs as simple as trees (with edge lengths) there is a lower bound of $\Omega(\log n \log\log n)$ bits on the label length [GKK+01]. As a consequence, our data structure cannot possibly be distributed as approximate distance labels (at least for graphs with edge lengths). Obviously, distance labels for planar graphs must be at least as long as those for trees — surprisingly enough they are not much longer.

Acknowledgments

Many thanks to Costas Busch for helpful discussions on [BLT07]. Thanks also to various anonymous reviewers for their comments and suggestions.

Work by KK was partially supported by the Japan Society for the Promotion of Science, a Grant-in-Aid for Scientific Research, by the C & C Foundation, by the Kayamori Foundation, and by an Inoue Research Award for Young Scientists.

Work by CS was partially supported by the Swiss National Science Foundation. CS also thanks Shay Mozes and Philip N. Klein for many fruitful discussions.

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 Ziyi 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 *4th European Symposium on Algorithms (ESA)*, pages 514–528, 1996.

[ACG12] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *44th Annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *25th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197, 2006. Details in LaBRI Research Report RR-1394-06.

[AG11] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *25th International Symposium on Distributed Computing (DISC)*, pages 404–415, 2011.

[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 *9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 33–41, 1998.

[AGMW10] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. *Theory of Computing Systems*, 47(4):837–855, 2010. Announced at SPAA 2007.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[Bak94] Brenda S. Baker. Approximation algorithms for NP–complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994. Announced at FOCS 1983.

[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.

[BGK+11] Yair Bartal, Lee-Ad Gottlieb, Tsvi Kopelowitz, Moshe Lewenstein, and Liam Roditty. Fast, precise and dynamic distance queries. In *22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 840–853, 2011.

[BGSU08] Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 609–621, 2008.

[BK06] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 591–602, 2006.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *26th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 61–70, 2007.

[BS06] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006. Announced at SODA 2004.

[Cab12] Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1–2):361–381, 2012. Announced at SODA 2006.

[Che95] Danny Ziyi Chen. On the all-pairs Euclidean short path problem. In *6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 292–301, 1995.

[CX00] Danny Ziyi Chen and Jinhui Xu. Shortest path queries in planar graphs. In *32nd ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

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

[DGNP10] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *3rd International Conference on Web Search and Web Data Mining (WSDM)*, pages 401–410, 2010.

[DGPW11] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *10th International Symposium on Experimental Algorithms (SEA)*, pages 376–387, 2011.

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

[DHK05] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 637–646, 2005.

[DHK11] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *43rd ACM Symposium on Theory of Computing (STOC)*, pages 441–450, 2011.

[DHM+09] Daniel Delling, Martin Holzer, Kirill Müller, Frank Schulz, and Dorothea Wagner. High-performance multi-level routing. In *The Shortest Path Problem: 9th DIMACS Implementation Challenge*, volume 74, pages 73–92. 2009.

[DHM10] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Bojan Mohar. Approximation algorithms via contraction decomposition. *Combinatorica*, 30(5):533–552, 2010. Announced at SODA 2007.

[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 *22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 151–165, 1996.

[DKT10] Zdenek Dvorak, Daniel Král, and Robin Thomas. Deciding first-order properties for sparse graphs. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 133–142, 2010.

[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.

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

[Epp03] David Eppstein. Dynamic generators of topologically embedded graphs. In *14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608, 2003.

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

[FJ90] Greg N. Frederickson and Ravi Janardan. Space-efficient message routing in c-decomposable networks. *SIAM Journal on Computing*, 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 *9th European Symposium on Algorithms (ESA)*, pages 476–487, 2001. Details in Research Report RR-1250-00.

[GKP95] Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph TSP. In *36th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 640–645, 1995.

[GKR04] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a Pez dispenser (or, routing issues in MPLS). *SIAM Journal on Computing*, 34(2):453–474, 2004. Announced at FOCS 2001.

[Gol07] Andrew V. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 88–102, 2007.

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

[Hag98] Torben Hagerup. Sorting and searching on the word RAM. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 366–398, 1998.

[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.

[HPM06] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006. Announced at SOCG 2005.

[HSW08] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13, 2008. Announced at ALENEX 2006.

[JHR96] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *5th International Conference on Information and Knowledge Management (CIKM)*, pages 261–268, 1996.

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

[Kle02] Philip Nathan Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 820–827, 2002.

[Kle05] Philip Nathan Klein. Multiple-source shortest paths in planar graphs. In *16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 146–155, 2005.

[Kle08] Philip Nathan Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. Announced at FOCS 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)*, pages 434–443, 2004.

[KMS12] Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. *arXiv*, abs/1208.2223, 2012.

[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.

[KPR93] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *25th ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993.

[KS98] Philip Nathan Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998. Announced at WADS 1993.

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

[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.

[MN07] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007. Announced at FOCS 2006.

[MS09] Manor Mendel and Chaya Schwob. Fast C-K-R partitions of sparse graphs. *Chicago Journal of Theoretical Computer Science*, pages 1–18, 2009.

[MS12] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–222, 2012.

[MZ07] Laurent Flindt Muller and Martin Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *15th European Conference on Algorithms (ESA)*, pages 657–668, 2007.

[New01] Mark E. J. Newman. Scientific collaboration networks. II. shortest paths, weighted networks, and centrality. *Physical Review E*, 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.

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

[PR10] Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup–Zwick bound. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

[PR11] Ely Porat and Liam Roditty. Preprocess, set, query! In *19th European Symposium on Algorithms (ESA)*, pages 603–614, 2011.

[PRT12] Mihai Patrascu, Liam Roditty, and Mikkel Thorup. Planning for fast connectivity updates. In *53rd IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, 2012.

[Rao87] Satish Rao. Finding near optimal separators in planar graphs. In *28th Symposium on Foundations of Computer Science (FOCS)*, pages 225–237, 1987.

[Rao92] Satish Rao. Faster algorithms for finding small edge cuts in planar graphs (extended abstract). In *24th ACM Symposium on Theory of Computing (STOC)*, pages 229–240, 1992.

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

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272, 2005.

[Som10] Christian Sommer. *Approximate Shortest Path and Distance Queries in Networks*. PhD thesis, The University of Tokyo, 2010.

[Som12] Christian Sommer. Shortest-path queries in static networks, 2012. submitted.

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

[SWZ02] Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *4th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 43–59, 2002.

[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. *Algorithms for Planar Graphs and Graphs in Metric Spaces*. PhD thesis, University of Copenhagen, 2010.

[WN10b] 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).

[WN10c] Christian Wulff-Nilsen. Min st-cut of a planar graph in $O(n \log\log n)$ time. *CoRR*, abs/1007.3609, 2010.

[WN12a] Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–208, 2012.

[WN12b] Christian Wulff-Nilsen. Approximate distance oracles with improved query time. *arXiv*, abs/1202.2336, 2012.

[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.

A Computing Sparse Neighborhood Covers of Planar Graphs

The algorithm of Busch, LaFortune, and Tirthapura [BLT07], computing a sparse neighborhood cover for some neighborhood radius r, runs in time $O(n \log n)$ for planar graphs.

The algorithm essentially recursively ($O(\log n)$ levels) separates G using shortest-path separators. Each of these separators can be found in linear time [LT79, Tho04]. Per level, the algorithm computes covers around these separator paths in a subgraph H. The sum of the nodes in all the subgraphs H of a level is at most $|V(G)| = n$.

Per subgraph H, the algorithm spends time $O(|V(H)|)$ as follows. We first find three separator paths Q_1, Q_2, Q_3 in time linear in $O(|V(H)|)$ [LT79, Tho04]. Each separator path Q_i is then decomposed into overlapping segments of length $2r$. A cluster around a segment Q_i^j is found by breadth-first search, exploring the $2r$–neighborhood of Q_i^j. Since each node v is in at most 3 clusters per path Q, v and the edges adjacent to v take part in at most $3 \cdot 3$ breadth-first searches per level.

In combination with algorithms of Rao [Rao87, Rao92] it may be possible to derive an $\tilde{O}(n)$–time algorithm that computes a constant-factor approximation for SparsestCut (uniform weights) in planar graphs.

Extension to Bounded-Genus Graphs. Note that the constructions of Busch, LaFortune, and Tirthapura [BLT07] can be extended to bounded-genus graphs using a result of Eppstein [Epp03].

PROPOSITION A.1. *For any graph G embedded into a surface of genus 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:* (i) *for each node v there is at least one subgraph G_i that contains all neighbors within distance r,* (ii) *each node v is contained in at most $2g+30$ subgraphs, and* (iii) *each subgraph has* radius *at most $\rho = 24r - 8$. Furthermore, given the embedding of G, such a sparse cover can be computed in time $O(gn \log n)$.*

Up to constants, this result is implied by the results in [AGMW10] — when using the result for minor-free graphs, the dependency on g would however become exponential as opposed to linear.

Proof. On a high level, the algorithm is actually rather simple, using existing subroutines as follows.

1. Following Eppstein's tree–cotree decomposition [Epp03], compute the top-level $2g$ paths (call this set $\mathcal{Q}$) in the shortest-path separators of Kawarabayashi, Klein, and Sommer [KKS11].
2. Compute covers for ("satisfy" the nodes on) each of these $O(g)$ paths $Q \in \mathcal{Q}$ (using the algorithm SHORTESTPATHCLUSTER of Busch, LaFortune, and Tirthapura [BLT07, Algorithm 1]),
3. Cut along each of these paths $Q \in \mathcal{Q}$; the resulting graphs are planar (see [Epp03]).
4. Compute covers for each of the remaining planar graphs (using the algorithm PLANARCOVER of Busch et al. [BLT07, Algorithm 5]).
5. Output the union of (2) and (4). □