Linear-Space Approximate Distance Oracles for Planar, Bounded-Genus, and Minor-Free Graphs

Ken-ichi Kawarabayashi[1], Philip N. Klein[2], and Christian Sommer[3]

[1] NII, Tokyo, Japan
[2] Brown U, Providence RI
[3] MIT, Cambridge MA

Abstract. A $(1+\epsilon)$-approximate distance oracle for a graph is a data structure that supports approximate point-to-point shortest-path-distance queries. The relevant measures for a distance-oracle construction are: space, query time, and preprocessing time.

There are strong distance-oracle constructions known for planar graphs (Thorup) and, subsequently, minor-excluded graphs (Abraham and Gavoille). However, these require $\Omega(\epsilon^{-1} n \lg n)$ space for n-node graphs.

We argue that a very low space requirement is essential. Since modern computer architectures involve hierarchical memory (caches, primary memory, secondary memory), a high memory requirement in effect may greatly increase the actual running time. Moreover, we would like data structures that can be deployed on small mobile devices, such as handhelds, which have relatively small primary memory.

In this paper, for planar graphs, bounded-genus graphs, and minor-excluded graphs we give distance-oracle constructions that require only $O(n)$ space. The big O hides only a fixed constant, independent of ϵ and independent of genus or size of an excluded minor. The preprocessing times for our distance oracle are also better than those for the previously known constructions. For planar graphs, the preprocessing time is $O(n \lg^2 n)$. However, our constructions have higher query times. For planar graphs, the query time is $O(\epsilon^{-2} \lg^2 n)$.

For bounded-genus graphs, there was previously no distance-oracle construction known other than the one implied by the minor-excluded construction, for which the constant is enormous and the preprocessing time is a high-degree polynomial. In our result, the query time is $O(\epsilon^{-2}(\lg n + g)^2)$ and the preprocessing time is $O(n(\lg n)(g^3 + \lg n))$.

For all these linear-space results, we can in fact ensure, for any $\delta > 0$, that the space required is only $1+\delta$ times the space required just to represent the graph itself.

1 Introduction

A $(1+\epsilon)$–*approximate distance oracle* for a graph is a data structure that supports point-to-point approximate distance queries. A distance-oracle construction for a family of graphs has three complexity measures:

- *preprocessing time:* time to build the data structure,
- *space:* how much space is occupied by the data structure, and
- *query time:* how long does it take for a query to be answered.

Each of these quantities might depend on the *stretch* parameter $1 + \epsilon$ (which is defined as the maximum ratio over all pairs of nodes of the query output divided by the length of a shortest path) as well as the size of the graph.

General graphs For general graphs, for stretch less than 2, no approximate distance oracle is known that achieves subquadratic space and sublinear query time. (In Section 2.1, we briefly survey work on general graphs.)

Restricted graph families The only known constructions that achieve $(1 + \epsilon)$ stretch are for restricted families of graphs: planar graphs [Tho04], minor-excluded graphs [AG06], and graphs of low doubling dimension [Tal04,HPM06,Sli07,BGK+10]. (In Section 2.2, we survey previously known results in this area.)

Fortunately, such graphs arise in applications, e.g. relating to road maps. Distance oracles can be used in finding nearby points of interest, in navigation and route-planning, and in algorithms for solving other optimization problems such as vehicle routing. A distance oracle was implemented as part of commercial software in 2001-2002 [Kle02].

Space requirements One obstacle to the widespread adoption of this technique may have been the space requirements of known distance oracles. Even the most compact distance oracle of Thorup [Tho04] requires $\Omega(\epsilon^{-1} n \lg n)$ space for n-node graphs. Even though the constant is quite modest, the storage required is rather large [MZ07].

Since modern computer architectures involve hierarchical memory (caches, primary memory, secondary memory), a high memory requirement in effect may greatly increase the actual query time. If the distance oracle could fit in cache, the query time could be much faster than if secondary or primary memory must be accessed. For smaller, less powerful mobile devices such as handhelds, the problem is exacerbated.

Our contribution In this paper, for every family of graphs for which a nontrivial $(1+\epsilon)$–approximate distance oracle is known (planar, bounded genus, H–minor-free, bounded doubling dimension), we give such a distance oracle that in addition requires only linear space.[4] In fact, for any $\delta > 0$, there is such a distance oracle whose space requirement is only $1 + \delta$ times the space required just to

[4] Our results for bounded-doubling-dimension graphs hold only for unit lengths.

store the graph itself; thus the overhead due to the distance oracle is in essence negligible.

We achieve this while increasing the query time by a factor that is almost proportional to the decrease in space.[5] For planar graphs, the query time of our oracle is $O(\epsilon^{-2} \lg^2 n)$. Although the query time for our constructions is higher than that for the superlinear-space constructions, the increase may be partly made up for by the decrease in actual time due to better memory performance (because of the memory hierarchy). The space/query-time tradeoff is tunable, so the construction can be adapted to a particular architecture.

The preprocessing time is also faster for our schemes than for the superlinear-space constructions.

Bounded-genus graphs For bounded-genus graphs, there was previously no distance-oracle construction known other than that implied by the minor-excluded construction, for which the constant is enormous and the preprocessing time is a high-degree polynomial. We give a more efficient construction tailored to graphs of genus g. For our linear-space oracle, the query time is $O(\epsilon^{-2}(\lg n + g)^2)$ and the preprocessing time is $O(n(\lg n)(g^3 + \lg n))$. (There is also an alternative construction with preprocessing time $O(n(\lg n)(g/\epsilon + \lg n))$.) We also provide an approximate distance oracle using space $O(n\epsilon^{-1}(\lg n + g))$ but having faster query time $O(g/\epsilon)$ (Theorem 3).

Summary of our results A summary of our results is given in Table 1.

Graph Class	Preprocessing	Query	
Planar Undirected	$O(n \lg^2 n)$	$O(\epsilon^{-2}(\lg n)^2)$	Theorem 1
Planar Directed	$O(n(\lg(nN))(\lg n)^3\epsilon^{-2})$	$O((\epsilon^{-1}(\lg n)(\lg(nN)))^2)$	Section 3.3
Reachability Oracle	$O(n \lg n)$	$O(\lg^2 n)$	Section 3.3
Genus g	$O(n(\lg n)(g^3 + \lg n))$	$O(\epsilon^{-2}(\lg n + g)^2)$	Theorem 2
H–minor-free	$O(poly(n, \epsilon))$	$O(\epsilon^{-2}(\lg n)^2)$	Theorem 4
α–doubling, unit lengths	$\epsilon^{-O(\alpha)}O(poly(n))$	$\epsilon^{-O(\alpha^2)} \cdot (\lg n)^{O(\alpha)}$	Theorem 5

Table 1. Time complexities of our linear-space $(1 + \epsilon)$–approximate distance oracles. N denotes the largest integer weight.

2 Previous work on approximate distance oracles

2.1 General and sparse graphs

Thorup and Zwick [TZ05] gave asymptotically almost optimal trade-offs for distance oracles for general undirected graphs, proving that for any graph and

[5] The product of space times query time for our oracle is $O(n\epsilon^{-2} \lg^2 n)$ while that same product is $O(n\epsilon^{-2} \lg n)$ for [Tho04]

for any integer k there is a $(2k-1)$–approximate distance oracle using space $O(kn^{1+1/k})$ and query time $O(k)$. They also prove that, if stretch strictly less than $2k+1$ is desired, then $\Omega(n^{1+1/k})$ bits of space are necessary. A slightly weaker lower bound holds for sparse graphs: Sommer, Verbin, and Yu [SVY09] prove that a distance oracle with stretch k and query time t requires space $n^{1+\Omega(1/(kt))}$ (up to poly-logarithmic factors). Tight with respect to this bound, the distance oracle of Mendel and Naor [MN07] has query time $O(1)$ and stretch $O(k)$ using space $O(n^{1+1/k})$. The oracle with the best stretch factor is by Pătraşcu and Roditty [PR10], who recently gave a 2–approximate distance oracle using space $O(n^{5/3})$ on sparse graphs. Distance oracles with stretch strictly less than 2 have not been achieved for general graphs.

2.2 Restricted graph classes: planar, excluded-minor, and bounded-doubling-dimension graphs

For restricted classes of graphs, better distance oracles are known and stretch $1+\epsilon$ can be achieved.

Planar graphs Thorup [Tho04] presents efficient $(1+\epsilon)$–approximate distance oracles for planar digraphs. (There is a slight improvement [Kle05] to the preprocessing time for one case.) Table 2 lists these results.

There are also many results on exact distance oracles for planar graphs. The best is that of Fakcharoenphol and Rao [FR06] and its subsequent improvements [Kle05,KMW10,MWN10] and variants [MS10,Nus10]. Better per-query time can be achieved by using more space [Cab06,MS10]. However, all these results require polynomial (but sublinear) query time. There are also results on special cases of planar graphs and special kinds of queries [DPZ95,DPZ00,DPZ91,CX00,KK03].

	Preprocessing	Space	Query	Reference
Directed	$O(n(\lg(nN))(\lg n)^3\epsilon^{-2})$	$O\left(n\cdot\epsilon^{-1}(\lg n)(\lg(nN))\right)$	$O(\lg\lg(nN)+\epsilon^{-1})$	[Tho04, Thm. 3.16]
Directed	$O(n(\lg n)^2(\lg(nN))\epsilon^{-1})$	$O\left(n\cdot\epsilon^{-1}(\lg n)(\lg(nN))\right)$	$O((\lg n)(\lg\lg(nN)+\epsilon^{-1}))$	[Tho04, Prop. 3.14]
Directed	$O(n(\lg n+\epsilon^{-1})(\lg n)(\lg(nN)))$	$O\left(n\cdot\epsilon^{-1}(\lg n)(\lg(nN))\right)$	$O((\lg n)(\lg\lg(nN)+\epsilon^{-1}))$	[Kle05, Sec. 7]
Undirected	$O(n(\lg n)^3\epsilon^{-2})$	$O\left(n\cdot\epsilon^{-1}\lg n\right)$	$O(\epsilon^{-1})$	[Tho04, Thm. 3.19]
Undirected	$O(n(\lg n)^2\epsilon^{-1})$	$O\left(n\cdot\epsilon^{-1}\lg n\right)$	$O(\epsilon^{-1}(\lg n))$	[Tho04, Implicit]

Table 2. Time and space complexities of $(1+\epsilon)$–approximate distance oracles for planar graphs on n nodes. N denotes the largest integer weight.

Excluded-minor graphs Abraham and Gavoille [AG06] extend Thorup's result to minor-free graphs. After a polynomial-time preprocessing step, point-to-point queries can be answered in time $O(\epsilon^{-1}\lg n)$ using a data structure of size $O(n\epsilon^{-1}\lg n)$.

Graphs of bounded doubling dimension Let Δ denote the aspect ratio (diameter divided by minimum distance) and let $\alpha = \lg_2 \lambda$ denote the doubling dimension. Har-Peled and Mendel [HPM06], improving upon earlier results by Talwar [Tal04] and Slivkins [Sli07], provide a $(1+\epsilon)$–approximate distance oracle (they term it *compact representation scheme*) using space $(1/\epsilon)^{O(\alpha)}n$ with query time $O(\alpha)$. Bartal et al. [BGK+10], extending [HPM06], recently gave a distance oracle with constant query time, at the cost of increasing the space consumption. We shift the trade-off in the reverse direction, increasing the query time while reducing the space requirement to linear (see Section B in the appendix for details).

3 Linear-space approximate distance oracle for planar graphs

We prove our main theorem. The description of the improved preprocessing algorithm can be found in its own section (Section 4).

Theorem 1. *For any undirected planar graph G with non-negative edge weights there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(\epsilon^{-2}\lg^2 n)$, linear space, and preprocessing time $O(n \lg^2 n)$.*

3.1 Review of Thorup's distance oracle

We briefly review a variant of Thorup's distance oracle for undirected graphs (using somewhat different terminology).[6]

There are two core ideas. The first is *approximately representing shortest paths that intersect a shortest path.* Let P be a shortest path in a graph G. A pair (p, v) of nodes where p is in P and v is in G is a *connection for v with respect to P*. A set $\mathcal{C}$ of such connections *covers v in G with respect to P* if, for every node p of P, there is a connection (p', v) in $\mathcal{C}$ such that

$$\operatorname{dist}(p, p') + \operatorname{dist}(p', v) \leq (1+\epsilon)\operatorname{dist}(p, v) \tag{1}$$

Let u, v be nodes of the input graph. Let Q be the shortest u-to-v path that intersects P. Suppose $\mathcal{C}$ is a set of connections that covers u and v with respect to P. Then it contains connections $(p, u), (p', v)$ such that

$$\operatorname{dist}(u, p) + \operatorname{dist}(p, p') + \operatorname{dist}(p', v) \leq (1+\epsilon)\operatorname{length}(Q) \tag{2}$$

Thorup gives an algorithm that, given a (mostly) planar graph G and a shortest path P, computes a set $\mathcal{C}$ of connections that covers all nodes of G and that has $O(\epsilon^{-1})$ connections per node v. In Section 4, we give an algorithm that

[6] This variant does not appear in [Tho04] but is an obvious simplification, analogous to that of his Proposition 3.14 (which applies to directed graphs) resulting in slower query times. Since our query time is slower anyway, this simplified variant suffices.

achieves a better[7] running time by covering only a subset of the nodes of G. The distance oracle involves storing with each node v the connections that cover v with respect to several shortest paths (and the distances associated with these connections). The storage required for v thus has size $O(\epsilon^{-1})$ times the number of such paths.

The second idea is *recursively decomposing a planar graph with shortest-path separators*.[8] This idea is based on a lemma in [LT79] stating that, for any spanning tree T in a planar graph in which every face is a triangle, there is a nontree edge e such that the unique simple cycle in $T \cup \{e\}$ is a balanced separator. The nodes of this separator comprise two paths in T.

The distance-oracle construction uses this lemma with T being a shortest-path tree to recursively decompose the input graph. The recursive decomposition defines a binary *decomposition* tree in which each node x is labeled by (i) a subgraph $G(x)$ of the input graph and (ii) the separator $S(x)$ used to decompose $G(x)$, if x is not a leaf. If x is the root, $G(x)$ is the input graph. If x has children y and z, removing the separator $S(x)$ from $G(x)$ results in two separated subgraphs, $G(y)$ and $G(z)$. If x is a leaf, $G(x)$ consists of one node.

Each input-graph node v is associated with some decomposition-tree node, namely, the leafmost node x whose subgraph includes v. We say that the ancestors of x are *relevant* to v. Thus each input-graph node v has $O(\lg n)$ relevant tree-nodes. The distance oracle assigns a label to v that consists of a set of connections; for each tree-node x relevant to v, for each of the two paths P comprising the separator $S(x)$, the distance oracle stores a set of connections that cover v in $G(x)$ with respect to P. It follows that the label of v has size $O(\epsilon^{-1} \lg n)$.

Next we show that these labels suffice to estimate point-to-point distances. We say that a tree-node x is *relevant* to a path Q if $S(x)$ contains a node of Q, and is the *most relevant* if x is the rootmost relevant tree node.

Lemma 1. *If x is the tree-node most relevant to Q then $G(x)$ contains Q.*

Let u, v be any pair of input-graph nodes, and let Q be the shortest u-to-v path. Let x be the tree-node most relevant to Q. Then $G(x)$ contains Q, and at least one of the paths comprising the separator $S(x)$, say P, intersects Q. It follows from (2) that the u-to-v distance is approximately

$$\operatorname{dist}(u, p) + \operatorname{dist}(p, p') + \operatorname{dist}(p', v) \tag{3}$$

for two nodes p, p' on P. To estimate the u-to-v distance, therefore, the following procedure suffices: for every tree-node x that is relevant to u and v, compute the minimum of (3) over connections (p, u) and connections (p', v) where p and p' belong to one of the two paths comprising $S(x)$. This takes time $O(\text{number of such connections})$. We review this process in Section 3.2 since in our case the situation is slightly more complicated.

[7] Our algorithm depends on G being wholly planar (as opposed to mostly planar as in [Tho04], which is the case for the variant we address.

[8] A similar but more involved such decomposition arose in [AGK+98].

3.2 Our compact distance oracle

Our linear-space construction draws on another kind of recursive decomposition using separators.

Definition 1 (Frederickson [Fre87]). *A* division *of a graph G is a partition of the edges of G 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 of an n-node planar graph G is a division of G into $O(n/r)$ subgraphs, called* regions*, with the following properties: (i) Each region contains $O(r)$ edges, and (ii) the number of boundary nodes in each region is at most $O(\sqrt{r})$.*

Note that there are $O(n/\sqrt{r})$ boundary nodes in total.

Lemma 2 (Frederickson [Fre87]). *A planar graph on n vertices can be divided into an r–division in $O(n \lg n)$ time.*

Using an r-division to get linear space Before carrying out the recursive decomposition with shortest-path separators, our preprocessing algorithm computes an r-division for $r = \ell^2$ (where ℓ is a parameter). Subsequently, connections (v, w) are only stored for those nodes v that are boundary nodes of the r-division. Since there are $O(n/\sqrt{r})$ boundary nodes, the connections and associated distances require storage $O((n\epsilon^{-1} \lg n)/\sqrt{r})$. We choose $\ell = \Theta(\epsilon^{-1} \lg n)$ so the total storage is $O(n)$.

An *s-to-t query* is handled as follows. First, compute shortest-path distances from s to all the nodes in s's region R_s. This takes $O(\ell^2)$ time [HKRS97]. At this point, the query algorithm has distances in the subgraph R_s from s to all the boundary nodes of R_s (and to t, if t is in R_s). There are $O(\ell)$ such boundary nodes. Similarly, compute shortest-path distances to t from all the nodes in t's region R_t, obtaining distances in the subgraph R_t to t from all the boundary nodes of R_t.

Let A, B be, respectively, the set of connections for boundary nodes of R_s, R_t. For each separator path P that has connections in A and B, the procedure described in Section 3.2 finds the shortest s-to-t path that enters P via a connection of A and leaves P via a connection of B. The time is linear in the number of such connections (see also [Tho04, Section 3.2.2] and [Tho04, Lemma 3.6]). Since each of the $O(\ell)$ boundary nodes of R_s and R_t has $O(\epsilon^{-1} \lg n)$ connections, the total time for these computations is $O(\ell\epsilon^{-1} \lg n)$.

Finally, return the minimum overall path-length (including the s-to-t distance within R_s, if t belongs to R_s). The total time for handling the query is $O(\ell^2 + \ell\epsilon^{-1} \lg n)$.

Details of the query algorithm We now explain how we find the shortest s-to-t path that enters P via a connection of A and leaves P via a connection of B. The method is a generalization of that in [Tho04, Sections 3.2.1 and 3.2.2].

For each connection (b, p) in A, b is a boundary node of R_s and we have $\text{dist}(s, b)$. For each connection (b, p) in B, b is a boundary node of R_t and we have $\text{dist}(t, b)$.

Let C be the sequence of all connections (s, p) and (t, p) in $A \cup P$, sorted according to the position of p on P. We use the following procedure.

initialize $m_s, m_t, d := \infty$
initialize $\hat{p} := p_0$
for each connection (p, b) in C in order,
    $m_s := m_s + \text{dist}(\hat{p}, p)$
    $m_t := m_t + \text{dist}(\hat{p}, p)$
    $\hat{p} := p$
    if b is a boundary node of R_s,
        $m_s := \min\{m_s, \text{dist}(s, b) + \text{dist}(b, p)\}$
    if b is a boundary node of R_t
        $m_t := \min\{m_t, \text{dist}(t, b) + \text{dist}(b, p)\}$
    $d := \min\{d, m_s + m_t\}$
return d

The procedure requires time $O(\text{number of connections considered})$. The procedure maintains the invariant that, after a node $\hat{p}$ of P has been considered in the loop, m_s is the length of the shortest s-to-$\hat{p}$ path of the form that goes via a boundary node b of R_s and a connection (b, p) and then travels along P from p to $\hat{p}$, where p appears before $\hat{p}$ on P. A similar statement holds for m_t. It follows that the value d returned by the procedure is the length of the shortest s-to-t path that travels in R_s to a boundary node b of R_s, then goes to P via a connection for b, then travels along P then leaves P via a connection for a boundary node b' of R_t then travels to t within R_t.

3.3 Extensions: distance oracles for planar digraphs and reachability oracles

Similar techniques (i.e. storing the connections for a subset of the nodes only) apply to more of Thorup's results based on shortest path separators [Tho04]. Instead of using a more sophisticated preprocessing algorithm that computes only the connections for the boundary nodes, we may compute the connections for all the nodes (using Thorup's preprocessing algorithms as black boxes) and then store only those for boundary nodes. By doing so, we obtain the following linear-space oracles. For directed planar graphs, there is a $(1 + \epsilon)$–approximate distance oracle with query time $O((\epsilon^{-1}(\lg n)(\lg(nN)))^2)$, where N denotes the largest integer weight. Furthermore, there is a reachability oracle with query time $O(\lg^2 n)$ (using [Tho04, Theorem 2.7]).

We exploit a similar black-box construction for minor-free graphs in Section 6.

4 Preprocessing algorithm for linear-space approximate distance oracle for planar graphs

Thorup's preprocessing algorithm for his undirected construction takes time $O(n\epsilon^{-2}\lg^3 n)$ (as stated in [Tho04, Theorem 3.19]). We give a preprocessing scheme for our construction that takes time $O(n\lg^2 n)$, independent of ϵ. We give details later, but here we observe that the factor $O(\epsilon^{-2}\lg n)$ speedup has three sources. (This explanation is aimed at readers familiar with Thorup's paper.)

First, since we are not aiming for a query time of $O(\epsilon^{-1})$, we can make do with a simpler preprocessing approach than the one underlying [Tho04, Theorem 3.19]; we use the approach that for directed graphs underlies [Tho04, Proposition 3.14]. The corresponding bound for undirected graphs is listed in Table 2 as "implicit".

Second, we only need to compute connections for a small subset of the nodes (the boundary nodes of the r-division). That in itself does not seem to permit an additional speedup using Thorup's method since his algorithm depends not on the number of connections stored but on the sizes of the graphs searched. Therefore, third, in addition we use another approach to finding connections, one based on the multiple-source shortest-path (MSSP) algorithm of Klein [Kle05] or that of Cabello and Chambers [CC07].

PREPROCESS(G_0)
  let B_0 be the set of boundary nodes of an r–division [Fre87]
  let T be a shortest-path tree
  compute recursive decomposition based on cycle separators of the form $T \cup \{e\}$
  for each nonroot node x of recursive-decomposition tree,
    for each path P_i $(i = 1, 2)$ comprising $S(x)$,
      compute connections for nodes of B_0 in $G(x)$ with respect to P_i

The last step, computing the connections for nodes of B_0 in $G(x)$ with respect to P_i, works on a graph $G'(x)$ obtained from $G(x)$ by cutting along P_i, duplicating the nodes and edges of P_i and creating a new face whose boundary consists of the two copies of P_i. This modification destroys paths that cross P_i but such paths are not needed since P_i is a shortest path. It has the advantage that, for each copy P of P_i, in $G'(x)$ all nodes of P lie on a common face.

For each copy P, there is a computation that selects connections (p, v) for specified nodes v with respect to that copy. The computation uses an algorithm called PATH(G, B, P) that takes time $O((|G|+\text{number of connections})\lg|G|)$ and selects $O(\epsilon^{-1})$ connections per node $v \in B$. Since there are two copies of two paths comprising $S(x)$, the last step of PREPROCESS selects $O(\epsilon^{-1})$ connections per node of B_0 in $G(x)$. Therefore the total number of connections for B_0 is $O(\epsilon^{-1}\lg n)$, and the total time is $O(n\lg^2 n + |B_0|\,\epsilon^{-1}\lg n)$, which is $O(n\lg^2 n)$.

Now we describe PATH(G, B, P). Let the nodes of P be $p_0 \ldots p_s$. First the algorithm computes

$$i(v) = \text{argmin}_i \text{dist}(p_i, v) \text{ and } d_v = \min_i \text{dist}(p_i, v).$$

These can be computed using a single-source shortest-path computation in the graph obtained by zeroing out the lengths of the edges of P.

For $i = 0, 1, \ldots, s$, let T_i denote the shortest-path tree rooted at p_i. For $i > 0$, let T_i' be the tree obtained from T_{i-1} by removing the parent edge of p_i and adding the edge $p_i p_{i-1}$, obtaining a p_i-rooted tree (not a shortest-path tree). For $i > 0$, let σ_i denote a sequence of edges whose insertion into T_i' (followed by the ejection of each corresponding parent edge) result in T_i.

Klein [Kle05] shows that each edge is inserted at most once, and gives an $O(|G| \lg |G|)$ algorithm (the *multiple-source shortest-path algorithm*) to compute these sequences. For each such inserted edge uv, the algorithm also computes the resulting change Δ_{uv} in the length of the root-to-v path in the tree. Cabello and Chambers [CC07] give a simplification of the multiple-source shortest-path algorithm and generalize it to bounded-genus in $O(g^2 |G| \lg |G|)$ time.

The algorithm PATH uses one of these algorithms to compute the sequences σ_i and the corresponding length changes Δ_{uv}.

4.1 The two phases

The remainder of PATH consists of two phases, FORWARD and BACKWARD. A connection (p_i, v) might be added by FORWARD if $i > i(x)$ and by BACKWARD if $i < i(x)$. We describe FORWARD. BACKWARD is symmetric.

4.2 The FORWARD phase

The algorithm FORWARD iterates through the nodes $p_0, \ldots, p_s$ of P, maintaining a tree T that is, in turn, $T_0, T_1, \ldots, T_s$. The tree T is represented using a dynamic-tree data structure [ABH+04,AHdLT05,Fre97,ST83,TW05]. A node-labeling is maintained: $\mu(v)$ is a quantity (discussed later) that is used to decide whether v needs a new connection. This labeling is represented implicitly, as is typical in dynamic trees, so as to support bulk updates. In this case (somewhat atypically), an update takes the form "add a quantity Δ to the label of every tree in the subtree rooted at u." Each update takes $O(\lg n)$ amortized time. In addition, searching for a node v that has $\mu(v) \leq 0$ takes $O(\lg n)$ time.

```
FORWARD(G, B, P):
    initialize T := T_0
    for every node v, initialize μ(v) := ∞
    for i = 0, 1, 2, ..., s:
        comment: T is rooted at p_i
⋆       for each node v ∈ B such that either i(v) = i or μ(v) ≤ 0 ,
            create a connection (p_i, v)
            set μ(v) := ε d_v
        if i < s,
            comment: now change the root...
            remove parent edge of p_{i+1} and add edge p_{i+1} p_i
            comment: now make the tree a shortest-path tree
```

† for each edge uv in the sequence σ_{i+1},
remove the current parent edge of v in T, and add uv
‡ forevery active node w in the v-rooted subtree of T,
$\mu(w) := \mu(w) + \Delta_{uv}$

The overall number of iterations of the loop in Step $\star$ is the number of connections added. The overall number of iterations of the loop in Step † is at most the number of edges, which is $O(|G|)$. Step ‡ can be done using a single bulk update in $O(\lg |G|)$ time. Consequently, the algorithm runs in time $O((|G| + \text{number of connections}) \lg |G|)$.

Now we show that the algorithm selects a covering set of connections (and that the set is small). At each moment in the execution of the algorithm, for each node v such that $\mu(v)$ is finite, let $\text{last}(v)$ denote the node p of P such that (p, v) was the most recently selected connection for v.

The μ *invariant* is: for every node v for which $\mu(v)$ is finite,

$$\mu(v) = \epsilon d_v - (\text{dist}(p_i, \text{last}(v)) + \text{dist}(\text{last}(v), v) - \text{dist}_T(p_i, v)) \tag{4}$$

Note that $\text{dist}(p_i, \text{last}(v)) + \text{dist}(\text{last}(v), v)$ is the length of the path that goes from the current root p_i to v via $\text{last}(v)$. When this length becomes significantly longer than $\text{dist}(p, v)$ (longer by ϵd_v), $\mu(v) \leq 0$ so the node v is included in the loop in Step $\star$, so the connection (p_i, v) is added. This shows that the connections added by FORWARD and BACKWARD cover each node $v \in B$.

To bound the number of connections, we follow Thorup in using the potential function $\Phi_v = \text{dist}(p_s, \text{last}(v)) + \text{dist}(\text{last}(v), v)$. Suppose that, at some execution of Step $\star$, $\mu(v) \leq 0$, so $\text{dist}(p_i, \text{last}(v)) + \text{dist}(\text{last}(v), v) - \text{dist}_T(p_i, v) \geq \epsilon d_v$. When a connection (p_i, v) is then added, $\text{last}(v)$ becomes p_i, so the potential function Φ_v is reduced by at least ϵd_v.

Initially $\Phi_v = \text{dist}(p_s, p_{i(v)}) + \text{dist}(p_{i(v)}, v)$. Throughout the phase, by the triangle inequality , $\Phi_v \geq \text{dist}(p_s, v)$. Again using the triangle inequality (and the fact that the graph is undirected), $\text{dist}(p_s, p_{i(v)}) \leq \text{dist}(p_s, v) + \text{dist}(p_{i(v)}, v)$, so $\Phi_v \geq \text{dist}(p_s, v) \geq \text{dist}(p_s, p_{i(v)}) - \text{dist}(p_s, v)$. Thus the total amount of reduction in Φ_v is at most $2\,\text{dist}(p_s, v)$. Since each reduction is by at least $\epsilon\,\text{dist}(r_{i(v)}, v)$, the total number of reductions (number of connections added by FORWARD after the initial one) is at most $\lceil 2\epsilon^{-1} \rceil$.

5 Approximate distance oracles for genus g graphs

Theorem 2. *For any undirected graph G embedded in a surface of Euler genus g, there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(\epsilon^{-2}(\lg n + g)^2)$, linear space, and preprocessing time $O(n(\lg n)(g^3 + \lg n))$. The oracle can also be constructed in time $O(n(\lg n)(g/\epsilon + \lg n))$.*

Our distance oracle for genus g graphs is based on separating shortest paths, as for planar graphs (see Section 3.1). Thorup [Tho04] proves that any planar graph can be recursively separated by three shortest paths. Abraham and

Gavoille [AG06] extend his result to minor-closed families, proving that any minor-free graph can be recursively separated by $O(1)$ shortest paths. Since bounded-genus graphs exclude minors, we could use their result to obtain a linear-space distance oracle. The constant in [AG06] however depends on the size of the minor in an unspecified way. In the following, we prove that genus g graphs can be recursively separated using at most $O(g)$ shortest paths. In fact, only the first separator consists of at most $2g$ paths, while lower levels can be separated using 3 paths. These smaller separators allow us to derive approximate oracles and labeling schemes with a dependency on g that is much lower than the corresponding dependency in the more general construction by Abraham and Gavoille [AG06]. More formally, we also prove the following.

Theorem 3 (fast distance queries for genus g graphs). *For any undirected graph G embedded in a surface of Euler genus g, there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(g/\epsilon)$, space $O(n(g+\lg n)/\epsilon)$, and preprocessing time $O(n(\lg n)^3\epsilon^{-2} + n(\lg n)g/\epsilon)$. The oracle can be distributed as a labeling scheme using $O((g+\lg n)/\epsilon)$ bits per node.*

In the following, we assume that G is embedded.

5.1 Overview

In the first step, we "cut" the genus g graph into planar subgraphs using the *tree-cotree decomposition* of Eppstein [Epp03], which decomposes a graph of genus g into planar graphs, separated by $2g$ paths from a tree T. We choose T to be a shortest-path tree.

Lemma 3 (Corollary of Eppstein [Epp03, Proof of Lemma 3.2]). *Any graph G of genus g on n nodes and m edges can be divided into planar subgraphs by a separator that consists of at most $2g$ shortest paths. Furthermore, these paths can be computed using a single-source shortest-path search in G plus $O(gm)$ time.*

At a high level, the theorems follow by combining Eppstein's lemma with the distance oracles for planar graphs (Thorup [Tho04] and Sections 3 and 4). Within the planar subgraphs, we use the distance oracles for planar graphs. In addition to computing the connections to the separator paths within each planar subgraph, we also need to compute the connections to the $O(g)$ tree-cotree decomposition paths. Note that the latter set of connections consists of paths that may pass through non-planar parts. To compute these, we may use either [CC07] or [Tho04, Lemma 3.12], depending on the values of g and ϵ.

5.2 Preprocessing algorithm

Within planar subgraphs To obtain the preprocessing and space bounds in Theorem 2, we use the preprocessing algorithm described in Section 4 with $r := \ell^2$, where $\ell = O(\epsilon^{-1}(\lg n + g))$. Since the number of connections per node is proportional to ℓ and since a $1/\sqrt{r}$–fraction of the nodes per subgraph lies on the boundary, the overall space consumption is linear.

To obtain the preprocessing and space bounds in Theorem 3, we use Thorup's algorithm [Tho04, Thm. 3.19] for $O(1/\epsilon)$ query time.

Connections to tree-cotree separator There are two options to compute these connections: (1) We may use [Tho04, Lemma 3.12] (which internally uses Thorup's $O(m)$ SSSP algorithm [Tho99,Tho00]). The lemma states that, for a path Q, we can compute an ϵ–covering set $\mathcal{C}(v, Q)$ for all nodes v in time $O(\epsilon^{-1} n(\lg n))$. (2) We may use the MSSP data structure for genus g graphs by Cabello and Chambers [CC07], which requires $O(g^2 n \lg n)$ preprocessing and then answers queries in time $O(\lg n)$. See planar preprocessing (Section 4) for details. The time required is $O(g^2 n \lg n + \text{number of connections} \cdot \lg n) = O(g^2 n \lg n)$.

We apply either lemma for the at most $2g$ paths of the tree-cotree decomposition. (For Theorem 3, the first option gives better asymptotic preprocessing time; for Theorem 2, the optimal choice depends on ϵ and g.)

5.3 Query algorithm

At query time, we can essentially use the same algorithm as for the planar case (Section 3.2 and [Tho04, Thm. 3.19]). The only difference to the planar case is that we also need to include the at most $2g$ paths separating the genus graph into planar subgraphs. To obtain the bound on the query time in Theorem 2, note that computing connections through these $\leq 2g$ separating paths can be done in time $O(\ell g/\epsilon)$ and that exploring both regions took time $O(\ell^2)$ (where $\ell = O(\epsilon^{-1}(\lg n + g))$).

6 Linear-space approximate distance oracle for H–minor-free graphs

Theorem 4. *For any minor H there is an integer $h = h(H)$ such that for any undirected H–minor-free graph G with n nodes and m edges there exists a $(1+\epsilon)$–approximate distance oracle with query time $O(h\epsilon^{-2} \lg^2 n)$, space $O(m)$, and polynomial preprocessing time.*

The proof of Theorem 4 is structurally the same as for the planar case. We again use r–divisions [Fre87], this time tailored to minor-free graphs using a separator algorithm by Kawarabayashi and Reed [KR10][9]. Due to space restrictions, more details are provided in Appendix A.

[9] Alternatively, Frederickson's algorithm could be combined with the separator algorithm by Alon, Seymour, and Thomas [AST94] to obtain an r–division with $O(|H|^{3/2}\sqrt{r})$ boundary vertices per region or with the separator algorithm by Reed and Wood [RW09] for $O(|H|^{3/2} 2^{(|H|^2+4)/2} r^{2/3})$ boundary vertices per region. See also [TMH09] for such modifications.

References

ABH+04. Umut A. Acar, Guy E. Blelloch, Robert Harper, Jorge L. Vittes, and Shan Leung Maverick Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. In *Proceedings of the Fifteenth ACM-SIAM Symposium on Discrete Algorithms*, pages 531–540, 2004.

AG06. Ittai Abraham and Cyril Gavoille. Object location using path separators. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 188–197, 2006. Details in LaBRI Research Report RR-1394-06.

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 *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California*, pages 33–41, 1998.

AHdLT05. Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005.

AST94. Noga Alon, Paul D. Seymour, and Robin Thomas. Planar separators. *SIAM Journal on Discrete Mathematics*, 7(2):184–193, 1994.

BGK+10. Yair Bartal, Lee-Ad Gottlieb, Tsvi Kopelowitz, Moshe Lewenstein, and Liam Roditty. Fast, precise and dynamic distance queries. *CoRR*, abs/1008.1480, 2010. To appear in SODA 2011.

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.

CC07. Sergio Cabello and Erin W. Chambers. Multiple source shortest paths in a genus g graph. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA*, pages 89–97, 2007.

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.

DPZ91. Hristo 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 Djidjev, Grammati E. Pantziou, and Christos D. Zaroliagis. Online and dynamic algorithms for shorted path problems. In *STACS*, pages 193–204, 1995.

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

Epp03. David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 599–608, 2003.

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.

Fre97. Greg N. Frederickson. A data structure for dynamically maintaining rooted trees. *Journal of Algorithms*, 24:37–65, 1997. Announced at SODA 1993.

GLNS08. Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms*, 4(1), 2008. Announced at SODA and ISAAC 2002.

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 J. Comput.*, 35(5):1148–1184, 2006. Announced at SOCG 2005.

KK03. Lukasz Kowalik and Maciej Kurowski. Short path queries in planar graphs in constant time. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 143–148, 2003.

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.

KMW10. Philip Nathan Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Transactions on Algorithms*, 6(2), 2010. Announced at SODA 2009.

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.

KR10. Kenichi Kawarabayashi and Bruce A. Reed. A separator theorem in minor-closed classes. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, 2010.

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

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.

MS10. Shay Mozes and Christian Sommer. Exact shortest path queries for planar graphs using linear space. *CoRR*, abs/1011.5549, 2010.

MWN10. Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *Algorithms - ESA 2010,*

18th Annual European Symposium, Liverpool, United Kingdom, September 6-8, 2010. Proceedings, 2010.

MZ07. Laurent Flindt Muller and Martin Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *Proceedings of the 15th annual European Conference on Algorithms*, pages 657–668, 2007.

Nus10. Yahav Nussbaum. Improved distance queries in planar graphs. *CoRR*, abs/1012.2825, 2010.

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.

RW09. Bruce A. Reed and David R. Wood. A linear-time algorithm to find a separator in a graph excluding a minor. *ACM Transactions on Algorithms*, 5(4), 2009.

Sli07. Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. *Distributed Computing*, 19(4):313–333, 2007. Announced at PODC 2005.

ST83. Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. Announced at STOC 1981.

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.

Tal04. Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 281–290, 2004.

Tho99. Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999. Announced at FOCS 1997.

Tho00. Mikkel Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35(2):189–201, 2000. Announced at STACS 1998.

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.

TMH09. Siamak Tazari and Matthias Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Applied Mathematics*, 157(4):673–684, 2009. Announced at WG 2008.

TW05. Robert Endre Tarjan and Renato Fonseca F. Werneck. Self-adjusting top trees. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 813–822, 2005.

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

A Linear-space approximate distance oracle for H–minor-free graphs

An *r–division* of an H–minor-free graph is a division into $\Theta(n/r)$ regions of $O(r)$ vertices each and $O(|H|\sqrt{r})$ boundary vertices each. (Note that the boundary is larger by a factor $|H|$ compared to the boundary in the planar case.)

Lemma 4 (Frederickson [Fre87], combined with Kawarabayashi and Reed [KR10]). *An H–minor-free graph on n vertices can be divided into an r–division in $O(n^2 \lg n)$ time.*

Proof (Proof of Theorem 4). The proof is a combination of three techniques: (i) Lemma 4, (ii) shortest path separators in minor-free graphs by Abraham and Gavoille [AG06], and (iii) Thorup's ϵ-covers (as described in Section 3.1).

Abraham and Gavoille [AG06] prove that any H–minor-free graph can be recursively separated by $k(H)$ shortest paths and that these paths can be found in polynomial time. We first compute these separator paths. We then compute an r–division for $r := (\ell \cdot |H|)^2$ as in Lemma 4. For all the nodes on the boundary, we compute connections to these shortest paths [Tho04, Lemma 3.12] (as in Section 5.2). We store $\ell = O(\epsilon^{-1} \lg n)$ connections per node (where ℓ depends on $|H|$ in an unspecified way) for $O(n/\ell)$ nodes. The total space requirement is thus $O(m)$ (without further dependencies on H).

At query time, given a pair (s, t), we first explore both regions R_s and R_t, respectively, using an SSSP search [TMH09] until all the boundary nodes of these two regions have been found. This step requires time $O(\ell^2)$ (hiding further dependencies on H stemming from [TMH09]). As described in Section 3.2, we can then merge clean and ordered covers in linear time [Tho04, Lemma 3.6].

B Linear-space approximate distance oracle for unit-length graphs with bounded doubling dimension

The distance oracles for planar, bounded-genus, and minor-free graphs heavily used the notion of *separators*. In the following, we show that separators are not the only way to obtain linear-space approximate distance oracles. Our linear-space approximate distance oracle for bounded-doubling-dimension graphs exploits the bounded-growth property.

The *aspect ratio* (also known as *spread*) of $P \subseteq V$, denoted by $\Delta(P)$, is the ratio of the diameter of P and the distance between the closest pair of nodes in P. It is well known that for any λ–doubling metric $\mathcal{M}$, any set of points $P \subset \mathcal{M}$ with aspect ratio at most $D \geq \Delta(P)$ satisfies $|P| \leq \lambda^{O(\lg D)}$.

We use this fact to obtain the following.

Theorem 5. *For any unit-length graph $G = (V, E)$ on $n = |V|$ nodes and $m = |E|$ edges with doubling dimension α and diameter Δ, and for any $\epsilon > 0$ there exists a $(1+\epsilon)$–approximate distance oracle using space $O(m)$ and query time $(\lg \Delta)^{O(\alpha)} \cdot (1/\epsilon)^{O(\alpha^2)}$.*

Note that (1) for unit-length graphs $\Delta = O(n)$ and thus the query time for constant α is $O(poly(\lg n, 1/\epsilon))$ and (2) our result also holds for unweighted geometric graphs such as those considered in [GLNS08]. Our approach extends to graphs with moderate edge lengths; the dependency on the largest weight is however polynomial and not logarithmic.

In our proof we use the following approximate distance labeling scheme.

Lemma 5 (Har-Peled and Mendel [HPM06, Proposition 6.10]). *For a metric with doubling dimension α and aspect ratio $\Delta = \Delta(V)$ and for any $\epsilon > 0$, there exists a $(1+\epsilon)$–approximate distance labeling scheme with label length $(1/\epsilon)^{O(\alpha)} \lg \Delta$ bits per node and query time $2^{O(\alpha)}$.*

Proof (Proof of Theorem 5).

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].

Let $G = (V, E)$ be a graph that allows for a $(1+\epsilon)$–approximate distance labeling scheme with label length $\ell = (1/\epsilon)^{O(\alpha)} \lg \Delta$ (Lemma 5). We store the distance labels for the nodes of an ℓ–dominating set, which requires total space $O(n)$. For each unlabeled node v, we also store its nearest labeled node $l(v)$.

At query time, given $s, t \in V$, we distinguish between "close" pairs (distance at most ℓ/ϵ) and "far" pairs (otherwise). For "close" pairs, we explore (using BFS) the ball $B(s)$ of radius $d = O(\ell/\epsilon)$ around s. If $t \in B(s)$, the exact distance can be returned (a "close" pair). Recall that for any λ–doubling metric $\mathcal{M}$, any set of points $P \subset \mathcal{M}$ with aspect ratio at most $D \geq \Delta(P)$ satisfies $|P| \leq \lambda^{O(\lg D)}$. For a unit-length graph $G = (V, E)$ with doubling dimension $\alpha = \lg_2 \lambda$, for any node $v \in V$ the number of nodes within distance d satisfies

$$|\{u : d_G(u, v) \leq d\}| \leq \lambda^{O(\lg d)}.$$

The number of edges within $B(s)$ is at most quadratic in the number of nodes. Exploring $B(s)$ using BFS thus requires time proportional to $\lambda^{O(\lg d)}$. For "far" pairs, we triangulate using $l(s)$ and $l(t)$, returning an approximate distance: The algorithm returns $\tilde{d}(u, v) = d_G(u, l(u)) + \mathcal{D}(\mathcal{L}(l(u)), \mathcal{L}(l(v))) + d_G(l(v), v)$, where $\mathcal{L}(w)$ denotes the label of w and $\mathcal{D}(\cdot, \cdot)$ denotes the decoding function of the labeling scheme. A simple calculation (using the triangle inequality and the fact that the distance $d(u, v)$ is at least ℓ/ϵ) yields that the query result $\tilde{d}(u, v)$ satisfies $\tilde{d}(u, v) \leq (1 + 7\epsilon) d(u, v)$ (for any $\epsilon \in (0, 1]$).