Noname manuscript No.
(will be inserted by the editor)

Efficient Dynamic Approximate Distance Oracles for Vertex-Labeled Planar Graphs

Itay Laish · Shay Mozes(ORCID: 0000-0001-9262-1821)

Received: date / Accepted: date

Abstract Let G be a graph where each vertex is associated with a label. A *vertex-labeled approximate distance oracle* is a data structure that, given a vertex v and a label λ, returns a $(1+\varepsilon)$-approximation of the distance from v to the closest vertex with label λ in G. Such an oracle is *dynamic* if it also supports label changes. In this paper we present three different dynamic approximate vertex-labeled distance oracles for planar graphs, all with polylogarithmic query and update times, and nearly linear space requirements. No such oracles were previously known.

Keywords Planar graphs · Approximate distance oracles · Vertex labels · Portals · ε-Cover

1 Introduction

Consider the following scenario. A 911 dispatcher receives a call about a fire and needs to dispatch the closest fire truck. There are two difficulties with locating the appropriate vehicle to dispatch. First, the vehicles are on a constant move. Second, there are different types of emergency vehicles, whereas the dispatcher specifically needs a fire truck. Locating the closest unit of a certain type under these assumptions is the *dynamic vertex-labeled distance query problem* on the road network graph. Each vertex in this graph can be annotated with a label that represents the type of the emergency vehicle currently located at that vertex. An alternative scenario where this problem is

This research was supported by the ISRAEL SCIENCE FOUNDATION (grant Nos. 794/13, 592/17).

An extended abstract of this work was presented in the 15th Workshop on Approximation and Online Algorithms (WAOA 2017), held in Vienna, Austria, September 2017.

Efi Arazi School of Computer Science
The Interdisciplinary Center Herzliya
E-mail: itaylaish@post.idc.ac.il, smozes@idc.ac.il

relevant is when one wishes to find a service provider (e.g., gas station, coffee shop), but different locations are open at different times of the day.

A data structure that answers distance queries between a vertex and a label, and supports label updates is called a *dynamic vertex-labeled distance oracle*. We model the road map as a planar graph, and extend previous results for the static case (where labels are fixed). We present oracles with polylogarithmic update and query times (in the number of vertices) that require nearly linear space.

We focus on approximate vertex-labeled distance oracles for a fixed parameter $\varepsilon > 0$. When queried, such an oracle returns at least the true distance, but not more than $(1 + \varepsilon)$ times the true distance. These are also known as stretch-$(1 + \varepsilon)$ distance oracles. Note that, in our context, the graph is fixed, and only the vertex labels change.

1.1 Related Work

1.1.1 Approximate vertex-to-vertex distance oracles

We outline related results, and refer the reader to an extensive survey due to Sommer [18]. For general graphs, Thorup and Zwick [20] presented for every $k \geq 2$ a stretch-$(2k-1)$ vertex-to-vertex distance oracle for undirected graphs with $O(kn^{1+\frac{1}{k}})$ space, and $O(k)$ query time. Their oracle can be constructed in $O(kmn^{1+\frac{1}{k}})$ time. Wulff-Nilsen [23] gave an oracle with similar space and query time, and $O(kn^{1+\frac{c}{k}})$ construction time. Here c is some universal constant.

For planar graphs, Thorup [19] presented a stretch-$(1+\varepsilon)$ distance oracle for directed planar graphs, for any $0 < \varepsilon < 1$. His oracle requires $O(\varepsilon^{-1} n \log n \log(nN))$ space and answers queries in $O(\log\log(nN) + \varepsilon^{-1})$ time. Here N denotes the ratio of the largest to smallest arc length. For undirected planar graphs Thorup presented an oracle that can be stored using $O(\varepsilon^{-1} n \log n)$ space, and can answer queries in $O(\varepsilon^{-1})$ time. Klein [10,11] independently described a stretch-$(1 + \varepsilon)$ distance oracle for undirected graphs with the same bounds, but faster preprocessing time. Kawarabayashi, Klein and Sommer [8], extended Thorup's result to other families of restricted graphs (e.g. minor free, bounded genus) and improved its space requirements to $O(n)$ in the cost of increasing the query time by a factor of $\varepsilon^{-1} \log^2 n$. Kawarabayashi, Sommer and Thorup [9] reduced the space dependency by a factor of $\varepsilon^{-1} \log n$, while keeping the query time of $O(\varepsilon^{-1})$. They also show an oracle for unweighted graphs that can be stored using $O(n)$ space, and has $O(\varepsilon^{-1})$ query time. Abraham, Chechik and Gavoille [2] presented a stretch-$(1+\varepsilon)$ distance oracle that supports both query and edge length updates in $\tilde{O}(n^{1/2})$ worst case time. Abraham et al. [1] later provided an oracle with polylogarithmic update and query time for planar graphs, when the edge lengths can only change within a predetermined ratio. In this work, we consider a different setting of updates, where the edge lengths are fixed, and only the vertex labels can change.

1.1.2 Approximate vertex-to-label distance oracles

The vertex-to-label query problem was introduced at ICALP'11 by Hermelin et al. [7]. For any $k \geq 2$, They presented a stretch-$(4k-5)$ distance oracle for undirected general (i.e. not necessarily planar) graphs with $O(kn^{1+\frac{1}{k}})$ space and query time $O(k)$. In a second result, they gave a dynamic label-to-vertex distance oracle that can handle label changes in sub-linear time, but with exponential stretch in terms of k, i.e., $(2 \cdot 3^{k-1} + 1)$. Chechik [3] later improved their results, and presented a stretch-$(4k-5)$ distance oracle that requires $\tilde{O}(n^{1+\frac{1}{k}})$ expected space, and supports queries in $O(k)$ time, and label changes in $O(n^{\frac{1}{k}} \log^{1-\frac{1}{k}} n \log\log n)$ time. Her oracle can be constructed using $O(kmn^{\frac{1}{k}})$ time.

The first result for the static vertex-to-label query problem for *undirected planar graphs* is due to Li et al. [12]. They described a stretch-$(1+\varepsilon)$ distance oracle that is based on Klein's results [10]. Their oracle requires $O(\varepsilon^{-1} n \log n)$ space, and answers queries in $O(\varepsilon^{-1} \log n \log \Delta)$ time. Here Δ is the hop-diameter of the graph, which can be $\Theta(n)$. Mozes and Skop [15], building on Thorup's oracle, described a stretch-$(1+\varepsilon)$ distance oracle for *directed planar graphs* that can be stored using $O(\varepsilon^{-1} n \log n \log(nN))$ space, and has $O(\log\log n \log\log nN + \varepsilon^{-1})$ query time.

Li et al. [12] considered the dynamic case, but their update method was trivial and takes $\Theta(n \log n)$ time in the worst case. Łącki et al. [14] presented a different dynamic vertex-to-label oracle for undirected planar graphs, in the context of computing Steiner trees. Their oracle requires $O(\sqrt{n} \log^2 n \log D \varepsilon^{-1})$ amortized time per update or query (in expectation), where D is the stretch of the metric of the graph (which could be nN). Their oracle however does not support changing the label of a specific vertex. Instead, they represent the labels in a forest, and support merging two labels by connecting two trees in the forest. Likewise, they support splitting labels by removing an edge from the forest, dividing a single tree into two trees.

To the best of our knowledge, our distance oracles are the first stretch-$(1+\varepsilon)$ vertex-to-label distance oracles with polylogarithmic query and update times, and the first that support directed planar graphs.

1.2 Our Results and Techniques

We present three approximate vertex-labeled distance oracles with polylogarithmic query and update times and nearly linear space and preprocessing times. Our update and construction times are expected amortized due to the use of dynamic hashing.[1] Our solutions differ in the tradeoff between query and update times. One solution works for directed planar graphs, whereas the other two only work for undirected planar graphs.

[1] We assume that a single comparison or addition of two numbers takes constant time.

We obtain our results by building on and combining existing techniques for the static case. All of our oracles rely on recursively decomposing the graph using shortest paths separators. Our first oracle for undirected graphs (Section 3) uses uniformly spaced connections, and efficiently handles them using fast predecessor data structures. The upshot of this approach is that there are relatively few connections. The caveat is that this approach only works when working with bounded distances, so a scaling technique [19] is required.

Our second oracle for undirected graphs (Section 5) uses the approach taken by Li et al. [12] in the static case. Each vertex has a different set of connections, which are handled efficiently using a dynamic prefix minimum query data structure. Such a data structure can be obtained using a data structure for reporting points in a rectangular region of the plane [21].

Our oracle for directed planar graphs (Section 4) is based on the static vertex-labeled distance oracle of [15], which uses connections for sets of vertices (i.e., a label) rather than connections for individual vertices. We show how to efficiently maintain the connections for a dynamically changing set of vertices using a bottom-up approach along the decomposition of the graph.

Our data structures support both queries and updates in polylogarithmic time. No previously known data structure supported both queries and updates in sublinear time. Table 1 summarizes the comparison between our oracles and the relevant previously known ones.

Table 1 Vertex-to-Label Distance Oracles Time Bound Comparison

		Query time	Update time
Li et al. [12]	U	$O(\varepsilon^{-1}\log n\log\Delta)$	$O(n\log n)$
Łącki et al. [14]	U	$O(\varepsilon^{-1}\sqrt{n}\log^2 n\log D)$	$O(\varepsilon^{-1}\sqrt{n}\log^2 n\log D)$
Sec. 3 (fast query)	U	$O(\varepsilon^{-1}\log n\log\log nN)$	$O(\varepsilon^{-1}\log n\log\log\varepsilon^{-1}\log nN)$
Sec. 5 (fast update)	U	$O(\varepsilon^{-1}\frac{\log^2(\varepsilon^{-1}n)}{\log\log(\varepsilon^{-1}n)})$	$O(\varepsilon^{-1}\log^{1.51}(\varepsilon^{-1}n))$
Mozes et al. [15]	D	$O(\varepsilon^{-1}+\log\log n\log\log nN)$	N/A
Sec. 4	D	$O(\varepsilon^{-1}\log n\log\log nN)$	$O(\varepsilon^{-1}\log^3 n\log nN)$

In the table above, D/U stands for Directed and Undirected graphs.

2 Preliminaries

2.1 Basic Concepts

Throughout the paper we use the term arc when dealing with directed graphs and the term edge when dealing with undirected graphs or when we wish to ignore the direction of an arc in a directed graph. Our algorithms are based on partitioning the input graph. For the sake of partitioning it is useful to regard the graph as undirected. For a directed graph G we let G' be the underlying undirected graph whose vertices $V(G') = V(G)$, and for every $u, v \in V(G)$,

$E(G')$ contains an edge uv if and only if $E(G)$ contains the arc uv or the arc vu. We say that P is an undirected path (resp., cycle or tree) in G if P is a path (resp., cycle or tree) in G'. We say that G is *connected* if for every $u, v \in V(G)$ there exists an u-to-v path in G'.

Let T be an undirected rooted tree. Let $A(\cdot)$ be a boolean property that is defined over the vertices of T. For every $v \in V(T)$, we say that a vertex u is *the root-most* vertex on the v-to-*root* path P in T that fulfills A, if u is the vertex closest to the root of T on the v-to-*root* path that fulfills A.

Let G be a directed graph. Let T be an undirected spanning tree of G rooted at some vertex $r \in V(G)$. For an edge uv not in T, the *fundamental cycle* of uv (with respect to T) is the undirected cycle composed of the r-to-u and r-to-v paths in T, and the edge uv.

Let $\ell : E(G) \to \mathbb{R}^+$ be a non-negative length function. Let N be the ratio of the maximum and minimum values of $\ell(\cdot)$. The length of a path P is $\Sigma_{e \in P} \ell(e)$. A shortest u-to-v path is a path of minimum length among those that start at u and end at v. We define the distance from u to v, denoted by $\delta_G(u, v)$, as the length of a shortest u-to-v path. We assume, only for ease of presentation, that shortest paths are unique. This assumption is only used when we present our algorithms and refer to *the* shortest path between two vertices. Our data structures do not require this assumption.

A path or cycle P is simple if every vertex is the endpoint of at most two arcs in P. For a simple path Q and a vertex set $U \subseteq V(Q)$ with $|U| \geq 2$, we define Q_U, the *reduction* of Q to U as a path whose vertices are U. Consider the vertices of U in the order in which they appear in Q. For every two consecutive vertices u_1, u_2 of U in this order, there is an arc u_1u_2 in Q_U whose length is the length of the u_1-to-u_2 sub-path of Q.

Let $\mathcal{L}$ be a set of labels. We say that a graph G is *vertex-labeled* if every vertex is assigned with a single label from $\mathcal{L}$. For a label $\lambda \in \mathcal{L}$, let S_G^λ denote the set of vertices in G with label λ. We define the distance from a vertex $u \in V(G)$ to the label λ by $\delta_G(u, \lambda) = \min_{v \in S_G^\lambda} \delta_G(u, v)$. If G does not contain the label λ, or λ is unreachable from u, we say that $\delta_G(u, \lambda) = \infty$.

Definition 1 For a fixed parameter $\varepsilon > 0$, a *stretch-*$(1 + \varepsilon)$ *vertex-labeled distance oracle* is a data structure that, given a vertex $u \in V(G)$ and a label $\lambda \in \mathcal{L}$, returns a distance d satisfying $\delta_G(u, \lambda) \leq d \leq (1 + \varepsilon)\delta_G(u, \lambda)$.

Definition 2 For fixed parameters $\alpha, \varepsilon > 0$, a *scale-*(α, ε) *vertex-labeled distance oracle* is a data structure that, given a vertex $u \in V(G)$ and a label $\lambda \in \mathcal{L}$, such that $\delta_G(u, \lambda) \leq \alpha$, returns a distance d satisfying $\delta_G(u, \lambda) \leq d \leq \delta_G(u, \lambda) + \varepsilon\alpha$. If $\delta_G(u, \lambda) > \alpha$, the oracle returns ∞.

A vertex-labeled distance oracle is *dynamic* if it also supports an update operation that, given a vertex v and a label λ, sets the label of v to be λ.

Definition 3 Let G be a directed graph. Let P be an undirected path in G. Let S be a set of vertex disjoint directed shortest paths in G. We say that P is *composed of* S if (i) each vertex of P is in some shortest path in S, and (ii)

for every path $Q \in S$, the undirected path corresponding to Q is a subpath of P.

Thorup shows that to obtain a stretch-$(1+\varepsilon)$ distance oracle for directed planar graphs, it suffices to show scale-(α, ε) oracles for so-called α-*layered* planar graphs. An α-layered graph is one equipped with an undirected spanning tree T such that each root-to-leaf path in T is composed of $O(1)$ shortest paths, each of length at most α. This is summarized in the following lemma:

Lemma 1 *([19, Sections 3.1,3.2,3.3]) Let G be a graph. Suppose that, for any $\alpha, \varepsilon' > 0$ and any α-layered minor H of G, one can construct, in $O(p(|H|, \varepsilon'))$ time[2] a scale-(α, ε') distance oracle with space bound $O(s(|H|, \varepsilon'))$ and query time $O(t(\varepsilon'))$ (here, p, s and t are arbitrary functions that do not depend on α). Then, one can construct, for any $\varepsilon > 0$, a stretch-$(1+\varepsilon)$ distance oracle for G with space $O(s(|G|, \frac{\varepsilon}{4}) \lg(|G|N))$ and query time $O(t(\frac{1}{4}) \log\log(|G|N) + t(\frac{\varepsilon}{4}))$. The construction time is $O(p(|G|, \frac{\varepsilon}{4}) \lg(|G|N))$.*

2.2 Recursive decomposition using shortest paths separators

The only properties of planar graphs that we use in this paper are the existence of shortest path separators (to be defined shortly), and the fact that single source shortest paths can be computed in $O(n)$ time in a planar graph with n vertices [6].

Definition 4 Let $G = (V, E)$ be a directed embedded planar graph. An undirected cycle C in G is a *balanced cycle separator* of G if the vertices of $V(G) \setminus V(C)$ can be partitioned into two subsets V_{int}, V_{ext}, each with at most $2|V(G)|/3$ vertices, and such that every path whose endpoints do not belong to the same subset intersects C. The subgraph of G induced by $V(C) \cup V_{int}$ (resp., $V(C) \cup V_{ext}$) is called the *interior* (resp., *exterior*) of C w.r.t. G. If, additionally, C is composed of a constant number of directed shortest paths, then C is called a *shortest path separator*.

Let G be a planar graph. We assume that G is triangulated since we can triangulate G with infinite length edges, so that distances are not affected. It is well known [13,19] that for any undirected spanning tree T of G, there exists a fundamental cycle C with respect to T that is a balanced cycle separator. The cycle C can be found in linear time. Note that, if T is chosen to be a shortest path tree, or if any root-to-leaf path of T is composed of a constant number of shortest paths, then the fundamental cycle C is a shortest path separator.

All of our distance oracles are based on a recursive decomposition of G using shortest path separators. If G is undirected (not necessarily α-layered), we can use any shortest path tree to find a shortest path separator in linear

[2] The lemma applies to any graph (not necessarily planar). The size of a graph H is defined as $|H| = |V(H)| + |E(H)|$. For planar graphs $|H| = O(|V(H)|)$.

time. Similarly, if G is directed and α-layered, then we can use the spanning tree G is equipped with to find a shortest path separator in linear time.

We now describe a recursive decomposition of G into subgraphs using shortest path separators until each subgraph has a constant number of vertices. We represent this decomposition by a binary tree $\mathcal{T}_G$. To distinguish the vertices of $\mathcal{T}_G$ from the vertices of G we refer to the former as *nodes*.

Each node r of $\mathcal{T}_G$ is associated with a subgraph G_r. The root of $\mathcal{T}_G$ is associated with the entire graph G. We sometimes abuse notation and equate nodes of $\mathcal{T}_G$ with their associated subgraphs. For each non-leaf node $r \in \mathcal{T}_G$, let C_r be the shortest path separator of G_r. Let Sep_r be the set of shortest paths C_r is composed of. The subgraphs G_{r_1} and G_{r_2} associated with the two children of r in $\mathcal{T}_G$ are the interior and exterior of C_r (w.r.t. G_r), respectively. The shortest path separator C_r belongs to both G_{r_1} and G_{r_2}. Note that the size of C_r is only bounded by the size of G_r. To guarantee that G_{r_i} $(i \in \{1, 2\})$ is smaller than G_r by a constant factor [19], C_r is replaced in G_{r_i} by the reduction of C_r to the vertices of C_r that have neighbors in $V(G_{r_i}) \setminus V(C_r)$. See [16] for a more comprehensive description of the recursive decomposition and the decomposition tree. Since we use this decomposition as a black box, we only highlight the following useful properties of $\mathcal{T}_G$. See Fig. 1 for an illustration.

- The depth of $\mathcal{T}_G$ is $O(\log n)$.
- The total size of all subgraphs corresponding to nodes at a specific depth of $\mathcal{T}_G$ is $O(n)$.
- The subgraph corresponding to each leaf of $\mathcal{T}_G$ has constant size.
- For each vertex v of G there exists at least one leaf r_v of $\mathcal{T}_G$ such that $v \in G_{r_v}$. If there is more than one such leaf, let r_v denote an arbitrary one.
- For each non-leaf node r of $\mathcal{T}_G$, Sep_r consists of a constant number of shortest paths.
- For every two vertices u, v of G, there exists some common ancestor r of r_u and r_v such that the shortest u-to-v path P in G is entirely contained in G_r, and such that P intersects some path in Sep_r.

Recall that we denote by $\mathcal{L}$ the set of labels of G. For $r \in \mathcal{T}_G$, let $\mathcal{L}_r$ be the restriction of $\mathcal{L}$ to labels that appear in G_r.

2.3 Connections sets

We now describe the connections sets, which are that basic building block used in our (and in many previous) distance oracles. Let u, v be vertices in G. Let Q be a path on the root-most separator (i.e., the separator in the node of $\mathcal{T}_G$ closest to its root) that is intersected by the shortest u-to-v path P. Let t be a vertex in $Q \cap P$. Note that $\delta_G(u, v) = \delta_G(u, t) + \delta_G(t, v)$. Therefore, if we stored for u the distance to every vertex on Q, and for v the distance from every vertex on Q, we would be able to find $\delta_G(u, v)$ by iterating over the vertices of Q, and finding the one minimizing the distance above. This, however, may require linear space since the number of vertices on Q might be

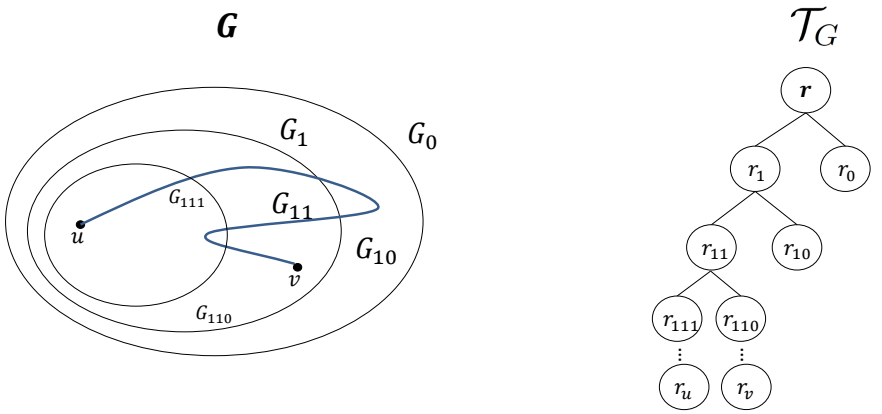


Fig. 1 An illustration of (part of) the recursive decomposition of a graph G using cycle separators, and the corresponding decomposition tree $\mathcal{T}_G$. The graph G is decomposed using a cycle separator into G_0, and G_1. Similarly, G_1 is decomposed into G_{10} and G_{11}, and G_{11} is decomposed into G_{110} and G_{111}. The node r is the root of $\mathcal{T}_G$ and is associated with $G_r = G$. Similarly, r_1 is associated with G_1, etc. The nodes r_u and r_v are the leaf nodes that contain u and v, respectively. The node r_1 is the root-most node whose separator is intersected by the shortest u-to-v path in G (indicated in blue). Hence, this path is fully contained in $G_{r_1} = G_1$.

$\theta(|V(G)|)$. Instead, we store the distances for a subset of Q. This set is called an (α, ε)-*covering connections set*.

Definition 5 ((α, ε)-covering connections set) [19, Section 3.2.1] Let $\varepsilon, \alpha > 0$ be fixed constants. Let G be a directed graph. Let Q be a shortest path in G of length at most α. For $u \in V(G)$ we say that $C_G(u, Q) \subseteq V(Q)$ is an (α, ε)-*covering connections set* from u to Q if and only if for every vertex t on Q s.t. $\delta_G(u, t) \leq \alpha$, there exists a vertex $q \in C_G(u, Q)$ such that $\delta_G(u, q) + \delta_G(q, t) \leq \delta_G(u, t) + \varepsilon\alpha$. The distances $\delta_G(u, q)$ are called *connections lengths*.

One defines (α, ε)-covering connections sets $C_G(Q, u)$ from Q to u symmetrically. We will use the term ε-covering connections set whenever α is obvious from the context. Various constructions of small covering connections sets are known for undirected and directed graphs. We will use different constructions in our different oracles, so we postpone their descriptions to the relevant sections.

3 An Oracle for Undirected Graphs (Variant with Faster Query)

In this section we describe a dynamic vertex labeled distance oracle for undirected planar graphs stated in the following theorem.

Theorem 1 *Let G be an undirected planar graph. There exists a stretch-$(1+\varepsilon)$ approximate dynamic vertex-labeled distance oracle that supports query in $O(\varepsilon^{-1}\log n\log\log nN)$ worst case time and updates in $O(\varepsilon^{-1}\log n\cdot\log\log\varepsilon^{-1}\log nN)$ expected amortized time. The construction time of that oracle is $O(\varepsilon^{-1}n\log n\cdot\log\log\varepsilon^{-1}\log nN)$ and it can be stored in $O(\varepsilon^{-1}n\log n\log nN)$ space.*

In fact, we only show a scale-(α,ε) distance oracle, which, by Lemma 1 implies the Theorem.

Let H be an undirected α-layered graph,[3] and let T be the associated spanning tree of H. We decompose H using shortest path separators w.r.t. T and obtain the decompostion tree $\mathcal{T}_H$. For every node $r\in\mathcal{T}_H$ and every shortest path $Q\in Sep_r$, we select a set $C_Q\subseteq V(Q)$ of ε^{-1} connections evenly spread intervals[4] along Q, each of length $\varepsilon\alpha$. Thus, for every vertex $t\in V(Q)$ there is a vertex $q\in C_Q$ such that $\delta_H(t,q)\le\varepsilon\alpha/2$.

For each $r\in\mathcal{T}_H$, for each shortest path $Q\in Sep_r$, for each $q\in C_Q$, we compute in $O(|H_r|)$ time a shortest path tree in H_r rooted at q using [6]. This computes the connection lengths $\delta_{H_r}(u,q)$, for all $u\in V(H_r)$.

Lemma 2 *Let $u\in V(H)$. For every ancestor node $r\in\mathcal{T}_H$ of r_u, and every $Q\in Sep_r$, C_Q is a ε-covering connections set from u to Q.*

Proof Let $t\in Q$. We need to show that there exist $q\in C_Q$ such that $\delta_{H_r}(u,t)\le\delta_{H_r}(u,q)+\delta_{H_r}(q,t)\le\delta_{H_r}(u,t)+\varepsilon\alpha$. Since $t\in Q$, there exists a vertex $q\in C_Q$ such that $\delta_H(q,t)\le\varepsilon\alpha/2$. Since H is undirected, the triangle inequality holds, so:

$$\delta_{H_r}(u,q)\le\delta_{H_r}(u,t)+\delta_{H_r}(t,q) \tag{1}$$
$$\delta_{H_r}(u,q)+\delta_{H_r}(t,q)\le\delta_{H_r}(u,t)+\delta_{H_r}(t,q)+\delta_{H_r}(t,q) \tag{2}$$
$$\delta_{H_r}(u,q)+\delta_{H_r}(t,q)\le\delta_{H_r}(u,t)+\varepsilon\alpha \tag{3}$$

By the triangle inequality we also have $\delta_{H_r}(u,t)\le\delta_{H_r}(u,q)+\delta_{H_r}(q,t)$, and the lemma follows. See Fig. 2 for an illustration. □

[3] The discussion of α-layered graphs in Section 2 refers to directed graphs, and hence also applies to undirected graphs.

[4] We assume that the endpoints of the intervals are vertices on Q, since otherwise one can add artificial vertices on Q without asymptotically changing the size of the graph.

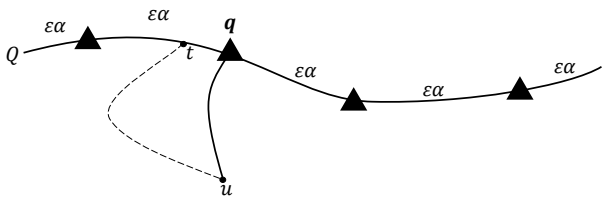


Fig. 2 Illustration of Lemma 2. Q is a shortest path in some separator, the connections of C_Q are marked by triangles. The solid u-to-q path reflects the shortest path from u to the connection q, and the dashed u-to-t path reflects the shortest path from u to t.

3.1 Warm Up: The Static Case

We start by describing our data structure for the static case with a single fixed label λ (i.e., each vertex either has label λ or no label at all). For every node $r \in \mathcal{T}_H$, let S_r^λ be the set of λ-labeled vertices in H_r. For every separator $Q \in Sep_r$, every vertex $q \in C_Q$, and every vertex $v \in S_r^\lambda$ let $\hat{\delta}_{H_r}(q,v) = k\varepsilon\alpha$ where k is the smallest integer such that $\delta_{H_r}(q,v) \leq k\varepsilon\alpha$. Thus, $\delta_{H_r}(q,v) \leq \hat{\delta}_{H_r}(q,v) \leq \delta_{H_r}(q,v) + \varepsilon\alpha$. Note that since in a scale-(α,ε) distance oracle we may assume distances are bounded by α, $k \leq \lceil \varepsilon^{-1} \rceil$. Let $L_r(q,\lambda)$ be the list of the distances $\hat{\delta}_{H_r}(q,v)$ for all $v \in S_r^\lambda$. Note that $L_r(q,\lambda)$ contains at most $\lceil \varepsilon^{-1} \rceil$ distinct values. We sort each list in ascending order. Thus, the first element of $L_r(q,\lambda)$ denoted by $first(L_r(q,\lambda))$ is at most $\varepsilon\alpha$ more than the distance from q to the closest λ-labeled vertex in H_r. We note that each vertex $u \in V(H)$ may contribute its distance to $O(\varepsilon^{-1}\log n)$ lists (one list for each of $O(\varepsilon^{-1})$ connections on each of $O(1)$ separator paths in each of the $O(\log n)$ nodes of $\mathcal{T}_H$ that contain u. Hence, we have $O(\varepsilon^{-1} n \log n)$ elements in total. Since H is an α-layered graph, the length of each Q is bounded by α. Hence, the universe of these lists can be regarded as non-negative integers bounded by $\frac{\alpha}{\varepsilon\alpha} = \varepsilon^{-1}$. Thus, these lists can be sorted in total $O(\varepsilon^{-1} n \log n)$ time.

3.1.1 Query(u,λ)

Given $u \in H$. We wish to find the closest λ-labeled vertex v to u in H. For each ancestor r of r_u, for each $Q \in Sep_r$, we perform the following search. We inspect for every $q \in C_Q$, the distance $\delta_{H_r}(u,q) + first(L_r(q,\lambda))$. We

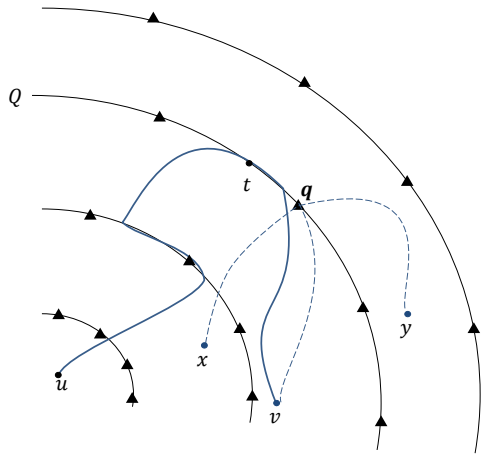


Fig. 3 Illustration of the query algorithm. The solid quarter-circles are shortest paths of separators in G. The vertices x, y and v have label λ, and v is the closest λ-labeled vertex to u. The path Q belongs to the root-most node r whose separator is intersected by the shortest u-to-λ path (solid blue). The vertices q and t on Q are as in the proof of Lemma 3. The connection q minimizes $\delta_H(u,q) + first(L_r(q,\lambda))$. The distances in $L_r(q,\lambda)$ are the lengths of the dashed paths.

also inspect the λ-labeled vertices in H_{r_u} explicitly. We return the minimum distance inspected. See Fig. 3 for an illustration.

Lemma 3 *The query algorithm runs in $O(\varepsilon^{-1}\log n)$ time, and returns a distance d such that $\delta_H(u,\lambda) \leq d \leq \delta_H(u,\lambda) + 2\varepsilon\alpha$.*

Proof Let v be the λ-labeled vertex closest to u in H. It is trivial that if the shortest path P from u to v does not leave $r_u = r_v$ the query algorithm is correct, since the distances in r_u are inspected explicitly. Otherwise, let r be the root-most node in $\mathcal{T}_H$ such that P intersects some $Q \in Sep_r$. Thus, P is fully contained in H_r. Let t be a vertex in $Q \cap P$. Since v is the closest λ-labeled vertex to u, it follows that v is also the closest λ-labeled vertex to t.

Let q be the vertex in C_Q closest to t. Thus $\delta_{H_r}(q,t) \leq \varepsilon\alpha/2$. By the triangle inequality we have the following two inequalities:

$$\delta_{H_r}(u,q) \leq \delta_{H_r}(u,t) + \delta_{H_r}(t,q) \leq \delta_{H_r}(u,t) + \varepsilon\alpha/2 \tag{4}$$

$$\delta_{H_r}(q,\lambda) \leq \delta_{H_r}(q,v) \leq \delta_{H_r}(t,v) + \delta_{H_r}(q,t) \leq \delta_{H_r}(t,v) + \varepsilon\alpha/2 \tag{5}$$

Summing the two inequalities we get $\delta_{H_r}(u,q) + \delta_{H_r}(q,\lambda) \leq \delta_{H_r}(u,v) + \varepsilon\alpha$. Since $first(L_r(q,\lambda) \leq \delta_{H_r}(q,\lambda) + \varepsilon\alpha$, we have $\delta_{H_r}(u,q) + first(L_r(q,\lambda) \leq \delta_{H_r}(u,v) + 2\varepsilon\alpha = \delta_H(u,\lambda) + 2\varepsilon\alpha$ (the last equality is by definition of v and r).

To prove the query time, observe that the height of $\mathcal{T}_H$ is $O(\log n)$. At any level of the decomposition we inspect the first element in $O(\varepsilon^{-1})$ lists, that is $O(\varepsilon^{-1}\log n)$ time. We also inspect a constant number of distances in r_u in constant time. □

We now generalize to multiple labels. Let $\mathcal{L}$ be the set of labels in H. Recall that, for $r \in \mathcal{T}_H$, we denote by $\mathcal{L}_r$ the restriction of $\mathcal{L}$ to labels that appear in H_r. For every label $\lambda \in \mathcal{L}_r$, every $Q \in Sep_r$ and every $q \in C_Q$, we store the list $L_r(q, \lambda)$. This does not affect the total size of our structure, since each vertex has one label, so it still contributes its distances to $O(\varepsilon^{-1} \log n)$ lists. The proof of Lemma 3 remains the same since each list contains distances to a single label.

Naively, we could store for every node r, every vertex q, and every label $\lambda \in \mathcal{L}$ the list $L_r(q, \lambda)$ in a fixed array of size $|\mathcal{L}|$. This allows $O(1)$-time access to each list, but increases the space by a factor of $|\mathcal{L}|$ w.r.t. the single label case. Instead, we use hashing. Each vertex q holds a hash table of the labels that contributed distances to q. For the static case, one can use perfect hashing [4] with expected linear construction time and constant query time. In the dynamic case, we will use a dynamic hashing scheme, e.g., [17], which provides query and deletions in $O(1)$ worst case, and insertions in $O(1)$ expected amortized time.

3.2 The Dynamic Case

We now turn our attention to the dynamic case. We wish to use the following method for updating our structure. When a node v changes its label from λ_1 to λ_2, we would like to iterate over all ancestors r of r_v in $\mathcal{T}_H$. For every $Q \in Sep_r$ and every $q \in C_Q$, we wish to remove the value contributed by v from $L_r(q, \lambda_1)$, and insert it to $L_r(q, \lambda_2)$. We must maintain the lists sorted, but do not wish to pay $O(\log n)$ time per insertion to do so. We will be able to pay $O(\log \log \varepsilon^{-1})$ per insertion/deletion by using a successor/predecessor data structure as follows.

For every $r \in \mathcal{T}_H$, $Q \in Sep_r$, and $q \in C_Q$, let $L_r(q)$ be the list containing *all distances* from all vertices in $V(H_r)$ to q sorted in ascending order. We note that since the distance for each specific vertex to q does not depend on its label, the list $L_r(q, \lambda)$ is a restriction of $L_r(q)$ to the λ-labeled vertices in H_r at all times.

During the construction of our structure we build $L_r(q)$, and, for every vertex v in H_r, we store for v its corresponding index in $L_r(q)$. We denote this index as $ID_q(v)$. Since $L_r(q)$ contains only $\lceil \varepsilon^{-1} \rceil$ distinct values, $ID_q(v)$ is an integer bounded by $\lceil \varepsilon^{-1} \rceil$ and all vertices u with the same $\hat{\delta}_{H_r}(q, u)$ share the same ID. We also store for q a single lookup table from the IDs to the corresponding distances. We note that v has $O(\varepsilon^{-1} \log n)$ such identifiers, and in total we need $O(\varepsilon^{-1} n \log n)$ space to store them.

Now, instead of using linked lists as before, we implement $L_r(q, \lambda)$ using a successor/predecessor structure over the universe $[1, ..., \varepsilon^{-1}]$ of the IDs. For an ordered set U (the universe) and a set $S \subseteq U$, the successor (respectively, predecessor) of $a \in U$ is the smallest (respectively, largest) element $b \in S$ such that b is greater (respectively, smaller) than a. A successor (predecessor) data structure for S is a data structure that supports insertion of elements to

S, deletion of elements from S, and answers successor (predecessor) queries for any $a \in U$. We use, e.g., y-fast tries [22], that, for the set $L_r(q, \lambda$, requires $O(|L_r(q, \lambda)|)$ space, and supports insertions, deletions, and queries in $O(\log\log\varepsilon^{-1})$ expected amortized time. It also supports minimum queries in $O(1)$ time in the worst case. (In fact, we only use insertions/deletions and minimum queries). In addition, we store for each ID present in $L_r(q, \lambda)$ a doubly linked list of the λ-labeled vertices with that ID. Each such vertex also stores a pointer to its node in the doubly linked list.

3.2.1 Query(u, λ)

The query algorithm remains the same as in the static case. For every ancestor r of r_u in $\mathcal{T}_H$, every $Q \in Sep_r$, and every connection $q \in C_Q$, we retrieve the minimal ID from $L_r(q, \lambda)$ using a minimum query in $O(1)$ time, and use the lookup table to get the actual distance between q and the vertex with that ID. The query time remains $O(\varepsilon^{-1}\log n)$.

3.2.2 Update

Assume that the vertex v changes its label from λ_1 to λ_2. For every ancestor r of r_v in $\mathcal{T}_H$, every $Q \in Sep_r$, and every $q \in C_Q$, we remove v from $L_r(q, \lambda_1)$ and insert v to $L_r(q, \lambda_2)$ as follows. We first remove v from the doubly linked list of $ID_q(v)$ in $L_r(q, \lambda_1)$. If v was the only node in the linked list we also remove $ID_q(v)$ from the predecessor data structure of $L_r(q, \lambda_1)$. Then, if $ID_q(v)$ is not present in $L_r(q, \lambda_2)$, we insert $ID_q(v)$ to the predecessor data structure of $L_r(q, \lambda_2)$. Finally, we add v to the doubly linked list of $ID_q(v)$ in $L_r(q, \lambda_1)$.[5]

Lemma 4 *The update time is $O(\varepsilon^{-1}\log n \cdot \log\log\varepsilon^{-1})$ expected amortized.*

Proof In each one of the $O(\log n)$ levels in $\mathcal{T}_H$, we perform $O(\varepsilon^{-1})$ insertions and deletions from successor/predecessor structures in $O(\log\log\varepsilon^{-1})$ expected amortized time per operation, and $O(\varepsilon^{-1})$ insertions and deletions in a linked list in $O(1)$ time per operation. Therefore the total update time is $O(\varepsilon^{-1}\log n\log\log n)$. If the set $\mathcal{L}_r$ changes for some $r \in \mathcal{T}_H$ as a result of the update, we must also update the hash table that handles the labels. This might cost an additional $O(1)$ expected amortized time per level, so is bounded by $O(\log n)$ expected amortized time in total. □

Lemma 5 *The data structure can be constructed in $O(\varepsilon^{-1}n\log n\cdot\log\log\varepsilon^{-1})$ expected time, and stored using $O(\varepsilon^{-1}n\log n)$ space.*

Proof We decompose H into $\mathcal{T}_H$, and compute the connection lengths in $O(\varepsilon^{-1}n\log n)$ time (to find the connection lengths for each connection q, we compute a shortest path tree rooted at q). We then build the lists $L_r(q)$ for

[5] Note that if one is only interested in reporting distances, and is not interested in being able to also trace back the shortest path, then the doubly-linked list can be replaced with a counter for the number of vertices represented by each ID in $L_r(q, \lambda)$.

every node $r \in \mathcal{T}_H$ and q on any $q \in Sep_r$. These lists contain $O(\varepsilon^{-1} n \log n)$ elements in the range $[1, ..., \varepsilon^{-1}]$ that is independent of both n and α. Hence we sort the lists in $O(\varepsilon^{-1} n \log n)$ time. We then use our update process on each $v \in V(H)$ and each ancestor r of r_v in $O(\varepsilon^{-1} \log n \cdot \log\log \varepsilon^{-1})$ expected amortized time for v. Hence, our construction time is $O(\varepsilon^{-1} n \lg n \log\log \varepsilon^{-1})$ in expectation. To see our space bound, we note that every v contributes a distance to $O(\varepsilon^{-1})$ lists at every ancestor r of r_v. Hence, there are $O(\varepsilon^{-1} n \log n)$ elements in total. Our successor/predecessor structures, and the hash tables require linear space in the number of elements stored. Thus, $O(\varepsilon^{-1} n \log n)$ space in total. □

Plugging this scale-(α, ε) distance oracle into Lemma 1 proves Theorem 1.[6]

4 An Oracle for Directed Graphs

In this section we describe a dynamic vertex labeled distance oracle for directed planar graphs.

Theorem 2 *For any directed planar graph and fixed parameter ε, there exists a $(1+\varepsilon)$ approximate vertex-labeled distance oracle that support queries in $O(\varepsilon^{-1} \log n \log\log nN)$ worst case and updates in $O(\varepsilon^{-1} \log^3 n \log nN)$ expected amortized time. This oracle can be constructed in $O(\varepsilon^{-2} n \log^5 n \log nN)$ expected time, and stored using $O(\varepsilon^{-1} n \log^3 n \log nN)$ space.*

As before, by Lemma 1, it suffices to show a dynamic vertex labeled scale-(α, ε) oracle. For simplicity we only describe an oracle that supports queries from a given label to a vertex. Vertex to label queries can be handled symmetrically.

Thorup proves that in any directed graph (not necessarily planar) there always exists an (α, ε)-covering connections set of size $O(\varepsilon^{-1})$:

Lemma 6 *[19, Lemma 3.4] Let $G, Q, \varepsilon, \alpha$ and u be as in definition 5. There exists an (α, ε)-covering connections set $C_G(u, Q)$ of size at most $\lceil 2\varepsilon^{-1} \rceil$. This set can be found in $O(|Q|)$ if the distances from u to every vertex on Q are given.*

Thorup further shows that (α, ε)-covering connections sets can be computed efficiently for the entire decomposition of a planar graph.

Lemma 7 *[19, Lemma 3.15] Let H be an α-layered graph. In $O(\varepsilon^{-2} n \log^3 n)$ time and $O(\varepsilon^{-1} n \log n)$ space one can compute and store a decomposition $\mathcal{T}_H$ of H using shortest path separators, along with (α, ε)-covering connections sets $C_H(u, Q)$ and $C_H(Q, u)$ for every vertex $u \in V(H)$, every ancestor node r of r_u in $\mathcal{T}_H$, and every $Q \in Sep_r$.*

[6] Formally, one needs to show that Lemma 1 holds for vertex-labeled oracles as well. We refer the reader to the detailed proof due to Mozes and Skop [16, Section 5.1].

To describe our data structure for directed graphs, we first need to introduce the concept of an ε-covering set from a *set of vertices* to a directed shortest path.

Definition 6 Let S be a set of vertices in a directed graph H. Let Q be a shortest path in H of length at most α. $C_H(S,Q) \subseteq V(Q) \times \mathbb{R}^+$ is an *ε-covering set* from S to Q in H if for every $t \in Q$ s.t. $\delta_H(S,t) \leq \alpha$, there exists $(q,\ell) \in C_H(S,Q)$ s.t. $\ell + \delta(q,t) \leq \delta_H(S,t) + \varepsilon\alpha$, and $\ell \geq \delta_H(S,q)$.

In the definition above we use ℓ instead of $\delta(S,q)$, and regard the connection lengths ℓ as a part of the covering set (compare to Definition 5). This is because we cannot afford to recompute exact distances as S changes. Instead, we store and use approximate distances ℓ.

Lemma 8 *Let H be a directed graph. Let Q be a shortest path in H of length at most α. For every set of vertices $S \subseteq V(H)$ there is an ε-covering set $C_H(S,Q)$ of size $O(\varepsilon^{-1})$.*

Proof We introduce a new apex vertex in H denoted by x. For every vertex v in S, we add an arc xv with length 0. Since the indegree of x is 0, Q remains a shortest path with length bounded by α. We apply Lemma 6 on x and Q, to get an ε-cover set $C_H(x,Q)$ of size $O(\varepsilon^{-1})$. Clearly, $C_H(x,Q)$ is an ε-covering set from S to Q, and the Lemma follows. □

Our construction relies on the following lemma.

Lemma 9 (Thinning Lemma) *Let H, S and Q be as in Lemma 8. Let $\{S_i\}_{i=1}^k$ be sets such that $S = \bigcup_{i=1}^k S_i$. For $1 \leq i \leq k$, let $D_H(S_i,Q)$ be an ε'-covering set from S_i to Q, ordered by the order of the vertices on Q. Then for every $\varepsilon > 0$, an $(\varepsilon+\varepsilon')$-covering set $C_H(S,Q)$ from S to Q of size $\lceil 2\varepsilon^{-1} \rceil$ can be found in $O(\varepsilon^{-1} + |\bigcup_{i=1}^k D_H(S_i,Q)|)$ time.*

Proof Let q_0 be the first vertex on Q. Let $\hat{Q}$ be the reduction of Q to $\{q_0\} \cup \{q : \exists \ell \text{ s.t. } (q,\ell) \in \bigcup_{i=1}^k D_H(S_i,Q)\}$. Let $\hat{H}$ be the auxiliary graph consisting of $\hat{Q}$ and an apex vertex x. For every pair (q,ℓ) in $\bigcup_{i=1}^k D_H(S_i,Q)$, we add to $\hat{H}$ an arc xq of length ℓ. Note that $\hat{H}$ has diameter bounded by α, and since the indegree of x is 0, Q is a shortest path in $\hat{H}$. Let $m = |\bigcup_{i=1}^k D_H(S_i,Q)|$. We compute the shortest distance from x to every other q in $\hat{H}$ explicitly by first relaxing (as in relaxations in Dijkstra's algorithm) all arcs adjacent to x, and then relaxing the arcs of Q starting from q_0 according to their order on Q. Constructing $\hat{H}$ and computing these distances therefore takes $O(m)$ time, since $|V(\hat{H})| = |E(\hat{H})| = O(m)$.

We apply Lemma 6 to x with ε and get an ε-covering set $C_{\hat{H}}(x,Q)$ of size $\lceil 2\varepsilon^{-1} \rceil$ from x to $\hat{Q}$. Note that $C_{\hat{H}}(x,Q)$ is a connections set of the vertex x. We convert $C_{\hat{H}}(x,Q)$ into a connections set for the set S as in Definition 6 by pairing each $q \in C_{\hat{H}}(x,Q)$ with $\delta_{\hat{H}}(x,q)$. From now on we treat $C_{\hat{H}}(x,Q)$

such a set of pairs. It remains to prove that $C_{\hat{H}}(x, Q)$ is an $(\varepsilon + \varepsilon')$-covering set from S to Q in H.

Let $t \in Q$. We show that there exists $(q, \ell) \in C_{\hat{H}}(x, Q)$ such that $\ell + \delta_H(q, t) \leq \delta_H(S, t) + (\varepsilon' + \varepsilon)\alpha$. We assume without loss of generality, that $\delta_H(S, t) = \delta_H(S_1, t)$. Since $D_H(S_1, Q)$ is an ε'-covering set from S_1 to Q in H, there exists $(q', \ell') \in D_H(S_1, Q)$ such that:

$$\ell' + \delta_H(q', t) \leq \delta_H(S_1, t) + \varepsilon'\alpha \tag{6}$$

Also, since $q' \in D_H(S_1, Q)$, it is also on $\hat{Q}$. Therefore there exists $(q, \ell) \in C_{\hat{H}}(x, Q)$ such that:

$$\ell + \delta_{\hat{H}}(q, q') \leq \delta_{\hat{H}}(x, q') + \varepsilon\alpha \leq \ell' + \varepsilon\alpha \tag{7}$$

Where the last inequality follows the fact that for every $(q^*, \ell^*) \in D_H(S_1, Q)$, $\delta_{\hat{H}}(x, q^*) \leq \ell^*$, and hence, $\delta_{\hat{H}}(x, q')$ is at most ℓ'.

$$\begin{aligned}
\delta_H(S, t) + \varepsilon'\alpha + \varepsilon\alpha &= \delta_H(S_1, t) + \varepsilon'\alpha + \varepsilon\alpha && (8)\\
&\geq \ell' + \delta_H(q', t) + \varepsilon\alpha && (9)\\
&\geq \ell + \delta_{\hat{H}}(q, q') + \delta_H(q', t) && (10)\\
&= \ell + \delta_H(q, q') + \delta_H(q', t) && (11)\\
&\geq \ell + \delta_H(q, t) && (12)
\end{aligned}$$

Here, (9) follows from inequality (6), (10) follows from inequality (7). □

Let H be a directed planar α-layered graph, equipped with a spanning tree T. For every fixed parameter $\varepsilon > 0$, let $\hat{\varepsilon} = \frac{\varepsilon}{8 \log n}$, and $\varepsilon^* = \frac{\varepsilon}{2}$. For every $1 \leq i \leq \log n$, let $\varepsilon_i = \frac{\varepsilon(\log n - i + 1)}{4 \log n}$. We denote the level (depth) of the root of $\mathcal{T}_H$ as 1, so the depth of the farthest leaf from the root is $\log n$ (ignoring rounding issues for simplicity).

The data structure for H consists of 3 types of connections sets at different levels of precision. We will use ε^*-covering sets for efficient queries. In order to support updates, we will be using the thinning lemma climbing up the $\log n$ levels of $\mathcal{T}_H$. Since we lose precision each time the thinning lemma is invoked we need to start with very precise $\hat{\varepsilon}$-covering sets at the leaves of $\mathcal{T}_H$ and maintain ε_i-covering sets at each level i of $\mathcal{T}_H$.

The data structure consists of the following:

- A decomposition tree $\mathcal{T}_H$ of H.
- For every $v \in V(H)$, every ancestor r of r_v in $\mathcal{T}_H$ and every $Q \in Sep_r$, $\hat{\varepsilon}$-covering sets $C_{H_r}(v, Q)$ and $C_{H_r}(Q, v)$
- For every $r \in \mathcal{T}_H$, λ in $\in \mathcal{L}_r$:
  - For every $Q \in Sep_r$ an ε^*-covering set $C^*_{H_r}(S^\lambda_r, Q)$, where S^λ_r is the set of all vertices with label λ in H_r.
  - For every ancestor t of r in $\mathcal{T}_H$ and every $Q \in Sep_t$, ε_i-covering sets $C_{H_t}(S^\lambda_r, Q)$ where i is the level of r in $\mathcal{T}_H$

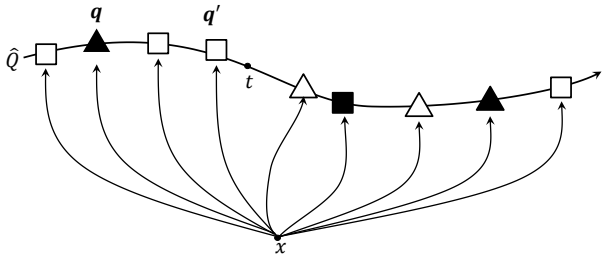


Fig. 4 Illustration of he auxiliary graph $\hat{H}$ in the proof of Lemma 9, when applied to the sets $S_1 = u$ and $S_2 = v$ (i.e. $k = 2$). The connections sets of S_1 and S_2 are indicated by the triangles and squares, respectively. The connections in the output set are indicated by a solid fill. The vertex t is a vertex on Q (not on $\hat{Q}$) that is closer to v than it is to u. Although t is covered by both q and q', its distance from x is better approximated via q'. The vertex q ε-covers q' w.r.t. the distances in $\hat{H}$ hence q' is not included in the output set. Since q ε-covers q', and q' ε'-covers t, it follows that t is $(\varepsilon + \varepsilon')$-covered by q.

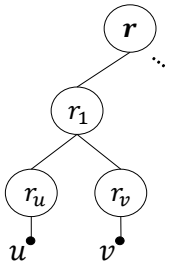
The first two items are computed by applying Lemma 7 with $\hat{\varepsilon}$ to H. The ε^*-covering sets are obtained by applying Lemma 9 (the thinning lemma) to the $\hat{\varepsilon}$-covering connections sets $C_{H_r}(v, Q)$ for all $v \in S_r^\lambda$, with $\varepsilon' = \hat{\varepsilon}$ and ε set to $\frac{\varepsilon}{4}$. We assume for the moment that the ε_i-covering sets are given. We defer the description of their construction (see the update procedure and the proof of Lemma 13).

Lemma 10 *The size of the data structure for an alpha-layered planar graph H with n vertices is $O(\varepsilon^{-1} n \log^3 n)$.*

Proof It is easy to see that the space is dominated by the total size of the ε_i-covering sets, so we only bound those. At each level of $\mathcal{T}_H$, the sum over $|\mathcal{L}_r$ for all nodes at that level of $\mathcal{T}_H$ is $O(n)$. For each of the $O(n)$ different λ's we store an $O(\varepsilon/\log n)$-covering set for $O(\log n)$ ancestral nodes in $\mathcal{T}_H$. Thus, the total size for each level is $O(n\varepsilon^{-1} \log^2 n)$, so $O(n\varepsilon^{-1} \log^3 n)$ for all levels. □

4.1 $Query(\lambda, u)$

The query algorithm is straightforward. For every ancestor r of r_u we find a query-connection (q, ℓ) in the ε^*-covering set of query connections $C^*_{H_r}(S_r^\lambda, Q)$, and a vertex connection t in the $\hat{\varepsilon}$-covering connections set $C_{H_r}(Q, u)$ that minimize the distance $\ell + \delta_{H_r}(q, t) + \delta_{H_r}(t, u)$. We also inspect the distance



	$\hat{\varepsilon}$-covering sets	ε^*-covering sets (used for query)	ε_i-covering sets (used for update)
r	N/A	From S^λ_G to every $Q \in Sep_r$	From S^λ_G to every $Q \in Sep_r$
r_1	N/A	From $S^\lambda_{r_1} = S_{r_u} \cup S_{r_v}$ to every $Q \in Sep_{r_1}$	From $S^\lambda_{r_1} = S_{r_u} \cup S_{r_v}$ to every $Q \in \{Sep_{r_1}, Sep_r\}$
r_u	N/A	From $S^\lambda_{r_u} = u$ to every $Q \in Sep_{r_u}$	From $S^\lambda_{r_u} = u$ to every $Q \in \{Sep_{r_u}, Sep_{r_1}, Sep_r\}$
r_v	N/A	From $S^\lambda_{r_v} = v$ to every $Q \in Sep_{r_v}$	From $S^\lambda_{r_v} = v$ to every $Q \in \{Sep_{r_v}, Sep_{r_1}, Sep_r\}$
u	From u to every $Q \in \{Sep_{r_u}, Sep_{r_1}, Sep_r\}$, and from every such Q to u.	N/A	N/A
v	From v to every $Q \in \{Sep_{r_v}, Sep_{r_1}, Sep_r\}$, and from every such Q to v.	N/A	N/A

Fig. 5 A summary of the connections-sets stored by the directed oracle. On the top, part of a decomposition tree of a graph is shown. The vertices u and v are the only λ labeled vertices in the graph. The table lists *all* the covering sets that are stored for the label λ.

to the λ-labeled vertices in r_u explicitly. We return the minimum distance inspected. To see that the query time is $O(\varepsilon^{-1} \log n)$, we note that for every one of the $O(\log n)$ ancestors of r_u we inspect $O(\varepsilon^{-1})$ distances on constant number of separators. Inspecting the distances in r_u itself takes constant time.

Lemma 11 $Query(\lambda, u) \leq \delta_H(\lambda, u) + \varepsilon\alpha$.

Proof Let r be the root-most node in $\mathcal{T}_H$ such that the shortest λ-to-u path P in H intersects some $Q \in Sep_r$. Let k be a vertex in $Q \cap P$. By the definition of the connections set $C^*_{H_r}(S^\lambda_r, Q)$, there exists (q, ℓ) such that

$$l + \delta_{H_r}(q, k) \leq \delta_{H_r}(S^\lambda_r, k) + \varepsilon^*\alpha \tag{13}$$

Also, there exists $t \in C_{H_r}(Q, u)$ such that

$$\delta_{H_r}(k, t) + \delta_{H_r}(t, u) \leq \delta_{H_r}(k, u) + \hat{\varepsilon}\alpha \leq \delta_{H_r}(k, u) + \varepsilon^*\alpha \tag{14}$$

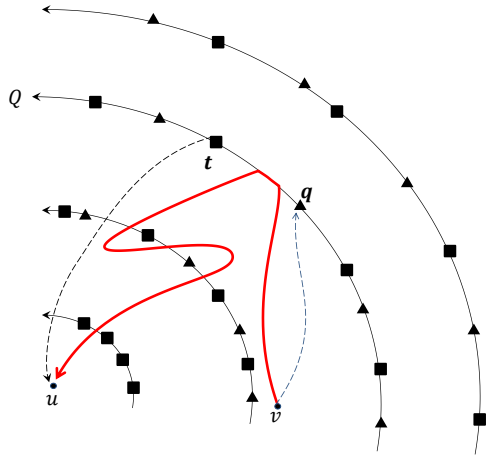


Fig. 6 Illustration of the query algorithm. The solid quarter-circles are shortest paths of separators in G. The vertex v is the closest λ-labeled vertex to u. The path Q belongs to the root-most node r whose separator is intersected by the shortest λ-to-u path (thick red). The connections of S_r^λ on Q are indicated by black triangles, the connections of u are indicated by black squares. The vertices q and t on Q are as in the proof of Lemma 11. The blue and black dashed lines are the shortest paths from v to q, and from t to u, respectively. These paths are used to generate the distance reported by the query algorithm.

We add the two to get

$$l + \delta_{H_r}(q,k)\delta_H(k,t) + \delta_{H_r}(t,u) \leq \delta_{H_r}(S_r^\lambda, k) + \delta_{H_r}(k,u) + \varepsilon^*\alpha + \varepsilon^*\alpha \quad (15)$$

$$l + \delta_{H_r}(q,t) + \delta_H(t,u) \leq \delta_{H_r}(S_r^\lambda, k) + \delta_{H_r}(k,u) + 2\varepsilon^*\alpha \quad (16)$$

$$l + \delta_{H_r}(q,t) + \delta_{H_r}(t,u) \leq \delta_{H_r}(S_r^\lambda, u) + \varepsilon\alpha \quad (17)$$

Clearly, $l + \delta_H(q,t) + \delta_H(t,u) \geq \delta_H(S_r^\lambda, u)$. Also, since P is fully contained in H_r, $\delta_{H_r}(S_r^\lambda, u) = \delta_H(S_r^\lambda, u)$, and the Lemma follows. □

4.2 Update

Assume that some vertex u changes its label from λ_1 to λ_2. For every ancestor r of r_u and every $Q \in Sep_r$, we would like to remove $C_{H_r}(u,Q)$ from $C_{H_r}(S_r^{\lambda_1}, Q)$, and combine $C_{H_r}(u,Q)$ into $C_{H_r}(S_r^{\lambda_2}, Q)$. While the latter is straightforward using Lemma 9, removing $C_{H_r}(u,Q)$ from $C_{H_r}(S_r^{\lambda_1}, Q)$ is more difficult. For example, if u was the closest λ_1 labeled vertex to every vertex on Q, it is possible that $C_{H_r}(u,Q) = C_{H_r}(S_r^{\lambda_1}, Q)$. In that case, we will have to rebuild $C_{H_r}(S_r^{\lambda_1}, Q)$ from the other $O(|V(H_r)|)$ vertices of $S_r^{\lambda_1}$. Instead of removing the connections of u, we will rebuild $C_{H_r}(S_r^{\lambda_1}, Q)$ bottom-up starting from the leaf node r_u.

We therefore start by describing how to update r_u. There is a constant number of vertices in r_u, and hence $|S_{r_u}^{\lambda_1}| = O(1)$. Let v_1, v_2,...v_k be the

vertices in $S^{\lambda_1}_{r_u}$. Recall that for every $1 \leq j \leq k$, v_j has an $\hat{\varepsilon}$-covering set $C_{H_t}(v_j, Q)$ of size $O(\hat{\varepsilon}^{-1})$ from v_j to Q, for *every ancestor* t of r_u in $\mathcal{T}_H$, and for every $Q \in Sep_t$. We apply the Thinning Lemma (Lemma 9) for each such t and Q on $\{C_{H_t}(v_j, Q)\}_{j=1}^{k}$ with $\varepsilon' = \hat{\varepsilon}$ and ε set to $\hat{\varepsilon}$. Lemma 9 yields a $2\hat{\varepsilon}$-covering set $C_{H_t}(S^{\lambda}_{r_u}, Q)$.

We next handle the ancestors r of r_u in $\mathcal{T}_H$ in bottom up order. Let x and y be the children of $r \in \mathcal{T}_H$. We first note that $H_r = H_x \cup H_y$ and hence, $S^{\lambda_1}_r = S^{\lambda_1}_x \cup S^{\lambda_1}_y$. Therefore, by Lemma 9, for every ancestor t of r, and every $Q \in Sep_t$, $C_{H_t}(S^{\lambda_1}_r, Q)$ can be obtained from $C_{H_t}(S^{\lambda_1}_x, Q) \cup C_{H_t}(S^{\lambda_1}_y, Q)$. Let i be the level of r in $\mathcal{T}_H$, and hence the level of x and y is $i+1$. Since t is an ancestor of r, it is also an ancestor of x and y. Hence, x (resp., y) stores an ε_{i+1}-covering set $C_{H_t}(S^{\lambda_1}_x, Q)$ (resp., $C_{H_t}(S^{\lambda_1}_y, Q)$). We apply Lemma 9 on $C_{H_t}(S^{\lambda_1}_x, Q)$ and $C_{H_t}(S^{\lambda_1}_y, Q)$ with $\varepsilon' = \varepsilon_{i+1}$ and $\varepsilon = 2\hat{\varepsilon}$ to get an $(\varepsilon_{i+1} + 2\hat{\varepsilon})$-covering set $C_{H_t}(S^{\lambda_1}_r, Q)$. The following lemma shows that $C_{H_t}(S^{\lambda_1}_r, Q)$ is an ε_i-covering set.

Lemma 12 *Let r be a node in level i in $\mathcal{T}_H$. For every ancestor t of r, and every $Q \in Sep_t$, $C_{H_t}(S^{\lambda_1}_r, Q)$ is an ε_i-covering set from $S^{\lambda_1}_r$ to Q.*

Proof We first recall that $\varepsilon_i = \frac{\varepsilon \log n - i + 1}{4 \log n}$ for every $1 \leq i \leq \log n$. We prove the lemma by induction on the level of r in $\mathcal{T}_H$. The base case is $i = \log n$, so r is a leaf. The connections sets of the leaf nodes are computed explicitly using Lemma 9, with ε and ε' set to $\hat{\varepsilon}$. Hence the product of the lemma is $2\hat{\varepsilon}$-covering sets.

$$2\hat{\varepsilon} = 2\frac{\varepsilon}{8 \log n} = \frac{\varepsilon}{4 \log n} = \varepsilon_{\log n} \tag{18}$$

For the inductive step, if r is a leaf, then the arguments from the base case apply. Otherwise, let x and y be the children of r. By the induction hypothesis, both x and y have ε_{i+1}-covering set from $S^{\lambda_1}_x$ and $S^{\lambda_1}_y$ to Q, respectively. The update procedure applies Lemma 9 on $C_{H_x}(S^{\lambda_1}_x, Q)$ and $C_{H_y}(S^{\lambda_1}_y, Q)$ with $\varepsilon' = \varepsilon_{i+1}$ and $\varepsilon = 2\hat{\varepsilon}$, so we get an $(\varepsilon_{i+1} + 2\hat{\varepsilon})$-covering set $C_{H_t}(S^{\lambda_1}_r, Q)$.

$$\varepsilon_{i+1} + 2\hat{\varepsilon} = \varepsilon\frac{\log n - (i+1) + 1}{4 \log n} + \varepsilon\frac{2}{8 \log n} = \varepsilon\frac{\log n - i + 1}{4 \log n} = \varepsilon_i \tag{19}$$

□

To finish the update process, we need to update the ε^*-covering sets that we use for queries. Let r be an ancestor node of r_u at level i in $\mathcal{T}_H$. By Lemma 12, for every $Q \in Sep_r$, we have an ε_i-covering set $C_{H_r}(S^{\lambda_1}_r, Q)$. Since $\varepsilon_i < \varepsilon^*$, $C_{H_r}(S^{\lambda_1}_r, Q)$ is also an ε^*-covering set. However, it is too large. We apply Lemma 9 on $C_{H_r}(S^{\lambda_1}_r, Q)$ with $\varepsilon' = \varepsilon_i$, and ε set to $\frac{\varepsilon}{4}$ to get $(\varepsilon_i + \frac{\varepsilon}{4})$-covering set. We note that since $\varepsilon_i \leq \frac{\varepsilon}{4}$ for every $1 \leq i \leq \log n$, we get that $\varepsilon_i + \frac{\varepsilon}{4} \leq 2\frac{\varepsilon}{4} \leq \frac{\varepsilon}{2} = \varepsilon^*$. Hence the output of Lemma 9 is the desired ε^*-covering set $C^*_{H_r}(S^{\lambda_1}_r, Q)$ of size $\lceil 2\varepsilon^{-1} \rceil$. We repeat the entire process for λ_2.

Lemma 13 *There exists a scale-(α, ε) distance oracle for directed α-layered planar graph, with worst case $O(\varepsilon^{-1} \log n)$ query time, and expected amortized $O(\varepsilon^{-1} \log^3 n)$ update time. The oracle can be constructed in $O(\varepsilon^{-2} n \log^5 n)$ time and stored using $O(\varepsilon^{-1} n \log^3 n)$ space.*

Proof The space requirement was established in Lemma 10. Since our update process only uses Lemma 9, we bound the update time by the running time of that Lemma. Since the running time of Lemma 9 is linear in the size of the input connections sets, we only need to bound the number of connections stored for r_u and all of its ancestors. We store for r_u a $\frac{\varepsilon}{4 \log n}$-connections set for constant number of separators for each one of the $O(\log n)$ ancestors of r_u. Hence, the number of connections stored for r_u is $O(\log n (\frac{\varepsilon}{4 \log n})^{-1}) = O(\varepsilon^{-1} \log^2 n)$. Since the number of connections stored for r_u dominates the number of connections stored for any strict ancestor of r_u, we get that the total number of connections stored for r_u and all its ancestors is $O(\varepsilon^{-1} \log^3 n)$.

We note that the connections of the vertices in r_u are only used when updating r_u, and for any other non-leaf node r, we only use the connections of its children. Thus, any connection of a node r is used at most twice. Once for updating a connections set of r's parent, and the second time when updating the ε^*-covering sets of r. Hence the total input size over all calls to Lemma 9 during the update of the label of u is at most twice the number of the connections stored for the ancestors of r_u, that is $O(\varepsilon^{-1} \log^3 n)$, and the update time follows.

Since we store for every $r \in \mathcal{T}_H$ and every $Q \in Sep_r$ a connections set for every $\lambda \in \mathcal{L}_r$, we use dynamic hashing as in Section 3. Hence, our update time is expected amortized.

Our query time is trivial and follows from the fact that we process $O(\log n)$ levels in $\mathcal{T}_H$, and in each we inspect $O(\varepsilon^{*-1})$ connections. That is $O(\varepsilon^{-1} \log n)$ worst case time.

We next analyze the construction time. By Lemma 7 with ε set to $\hat{\varepsilon}$, all connections sets for all leaves of $\mathcal{T}_H$ can be computed in $O(\varepsilon^{-2} n \log^5 n)$ and require $O(\varepsilon^{-1} n \log^2 n)$ space. We construct the connections sets $C_{H_r}(S_r^\lambda, Q)$ for all $r \in \mathcal{T}_H$, $Q \in Sep_r$ and $\lambda \in H_r$ by applying the update process for each vertex $v \in V(H)$. This takes $O(\varepsilon^{-1} \log^3 n)$ expected amortized time per vertex, and $O(\varepsilon^{-1} n \log^3 n)$ expected time in total. This is dominated by the $O(\varepsilon^{-2} n \log^5 n)$ term. □

Applying Lemma 1 with the scale-(α, ε) oracle of Lemma 13 proves Theorem 2.

5 Another Oracle for Undirected Graphs (Variant with Faster Update)

In this section we describe another oracle for undirected planar graphs, which is based on the static oracle of Li et al. [12].

Theorem 3 *For any undirected planar graph and fixed parameters ε, γ, there exists a stretch-$(1+\varepsilon)$ vertex-labeled distance oracle that approximates distances in $O(\varepsilon^{-1}\frac{\log n \log(\varepsilon^{-1}n)}{\log\log(\varepsilon^{-1}n)})$ worst casetime, and supports updates in $O(\varepsilon^{-1}\log n \log^{\frac{1}{2}+\gamma}(\varepsilon^{-1}n))$ expected amortized time. This data structure can be constructed using $O(n\log^2 n + \varepsilon^{-1} n \log n \log^{\frac{1}{2}+\gamma}(\varepsilon^{-1}n))$ expected amortized time and stored using $O(\varepsilon^{-1} n \log n)$ space.*

Both Thorup [19, Lemma 3.19] and Klein [10] independently presented efficient vertex-vertex distance oracles for undirected planar graph that use connections sets. Klein later improved the construction time [11]. They show that, in undirected planar graphs, one can avoid the scaling approach that uses α-layered graphs. Instead, there exist connections sets that approximate distance with $(1+\varepsilon)$ multiplicative factor rather than $\varepsilon\alpha$ additive factor. We use the term *portals* [11] to distinguish these types of connections from the previous ones.

Definition 7 Let G be an undirected planar graph, and let Q be a shortest path in G. For every vertex $v \in V(G)$ we say that a set $C_G(v,Q) \subseteq V(Q)$ is an ε-covering set of *portals* if and only if, for every vertex t on Q there exist a vertex $q \in C_G(v,Q)$ such that: $\delta_G(v,q) + \delta_G(q,t) \leq (1+\varepsilon)\delta_G(v,t)$

We use a recursive decomposition $\mathcal{T}_G$ with shortest path separators. For every vertex $v \in V(G)$ and for every ancestor r of r_v in $\mathcal{T}_G$, for every Q in Sep_r, we use Klein's algorithm [11] to select a portal set $C_{G_r}(u,Q)$ of size $O(\varepsilon^{-1})$ efficiently . We cannot use the lists of Section 3 because there may be too many portals, and we cannot use the thinning lemma (Lemma 9) of Section 4 because its proof uses a directed construction, and hence, cannot be applied in undirected graphs. Instead, we take the approach used by Li et al. for the static vertex-labeled case [12]. We work with all portals of vertices with the appropriate label, and find the closest one using dynamic prefix/suffix minimum queries.

Definition 8 (Dynamic prefix minimum data structure) A dynamic prefix minimum data structure is a data structure that maintains a set A of pairs in $[1,n] \times \mathbb{R}$, under insertions, deletions, and prefix minimum queries (PMQ) of the following form: given $l \in [1,n]$ return a pair $(x,y) \in A$ s.t. $x \in [1,l]$, and for every other pair $(x',y') \in A$ with $x' \in [1,l]$, $y \leq y'$.

Suffix minimum queries (SMQ) are defined analogously. Let $PMQ(A,l)$ and $SMQ(A,l)$ denote the result of the corresponding queries on the set A and l.

We assume that for every $u, v \in V(G_r)$, $C_{G_r}(u,Q) \cap C_{G_r}(v,Q) = \emptyset$. This is without loss of generality, since if x is a portal of vertices $v_0, ..., v_k$, we can split x to k copies. Since the size of $\cap C_{G_r}(v_i,Q)$ is $O(\varepsilon^{-1})$ for every v_i, this does not increase $|G|$ by more than a factor of ε^{-1}. See Fig. 7.

To describe our data structure, we first need the following definitions. Let $Q \in Sep_r$ for some $r \in \mathcal{T}_G$. Let $q_0, ..., q_k$ be the vertices on Q by their order along Q. G is undirected, so the direction of Q can be chosen arbitrarily. For

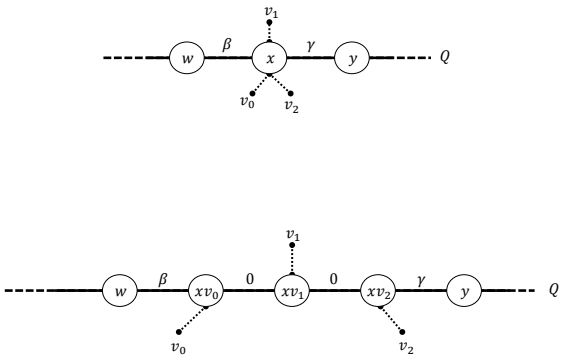


Fig. 7 Illustration of the reduction to unique portals. Above, the path Q with the portal x that is used by v_0, v_1, and v_2. Below, x was replaced by xv_0, xv_1, and xv_2, inner connected with zero length edges. Here, xv_0, xv_1, xv_2 are the portals of v_0, v_1, and v_2 respectively. Note that this reduction does not introduce new paths in the graph, nor changes the distance along Q.

every $0 \leq j \leq k$, let $h(q_j)$ denote the distance from q_0 to q_j on Q. We note that since Q is a shortest path in G, $h(q_i) = \delta_G(q_0, q_j)$.

For every $r \in \mathcal{T}_G$, for every $Q \in Sep_r$ and for every $\lambda \in \mathcal{L}_r$ we maintain a dynamic prefix minimum data structure $Pre_{Q,\lambda}$ over $\{(j, -h(q_j) + \delta_{G_r}(q_j, \lambda))\}_{j=0}^{k}$. We similarly maintain a dynamic sufix minimum data structure $Suf_{Q,\lambda}$ over $\{(j, h(q_j) + \delta_{G_r}(q_j, \lambda))\}_{j=0}^{k}$.

To Summarize, our data structure consists of:

- For every vertex $v \in V(G)$ and for every ancestor r of r_v in $\mathcal{T}_G$, for every Q in Sep_r the portal set $C_{G_r}(u, Q)$.
- For every $r \in \mathcal{T}_G$, for every $Q \in Sep_r$ and for every $\lambda \in \mathcal{L}_r$ the data structures $Pre_{Q,\lambda}$ and $Suf_{Q,\lambda}$.

5.1 $Query(u, \lambda)$

For every ancestor r of r_u in $\mathcal{T}_G$, every $Q \in Sep_r$, and every $q_j \in C_{G_r}(u, Q)$ we wish to find the index i that minimizes $\delta_{G_r}(u, q_j) + \delta_{G_r}(q_j, q_i) + \delta_{G_r}(q_i, \lambda)$. Observe that for $i \leq j$, $\delta_{G_r}(q_j, q_i) = h(j) - h(i)$, while for $i \geq j$, $\delta_{G_r}(q_j, q_i) = h(i) - h(j)$. We therefore find the optimal $i \leq j$ and $i \geq j$ separately. Note that $\min_{i \leq j}(\delta_{G_r}(u, q_j) + \delta_{G_r}(q_j, q_i) + \delta_{G_r}(q_i, \lambda)) = \delta_{G_r}(u, q_j) + h(j) + PMQ(Pre_{Q,\lambda}, j)$. Similarly, we handle the case where $i \geq j$ using $SMQ(Suf_{Q,\lambda}, j)$. Thus, we have two queries for each portal of u. We also compute the distance from u to λ in r_u explicitly. We return the minimum distance computed.

Lemma 14 *The query algorithm returns a distance d such that $\delta_G(u,\lambda) \leq d \leq (1+\varepsilon)\delta_G(u,\lambda)$*

Proof The proof of correctness of our algorithm is essentially the same as in [12, Lemma 1]. We adapt it to fit our construction. Let v be the closest λ-labeled vertex to u in G. If the shortest u-to-v path P does not leave $r_u = r_v$ the algorithm is correct, since the distance in r_u is computed expilicitly. Otherwise, let r be the root-most node in $\mathcal{T}_G$ such that P intersects some $Q \in Sep_r$. Let t be a vertex on $P \cap Q$. There exists $q_j \in C_{G_r}(u,Q)$ and $q_i \in C_{G_r}(v,Q)$ such that:

$$\delta_{G_r}(u,q_j) + \delta_{G_r}(q_j,t) \leq (1+\varepsilon)(\delta_{G_r}(u,t)) \tag{20}$$
$$\delta_{G_r}(v,q_i) + \delta_{G_r}(q_i,t) \leq (1+\varepsilon)(\delta_{G_r}(v,t)) \tag{21}$$

We add the two inequalities and use the triangle inequality to get:

$$\delta_{G_r}(u,q_j) + \delta_{G_r}(q_j,q_i) + \delta_{G_r}(v,q_i) \leq (1+\varepsilon)(\delta_{G_r}(u,v)) \tag{22}$$

If $i \leq j$, then $PMQ(Pre_{Q,\lambda}, j) \leq \delta_{G_r}(q_i,\lambda) - h(q_i) \leq \delta_{G_r}(v,q_i) - h(q_i)$. Thus,

$$Query(u,\lambda) \leq \delta_{G_r}(v,q_i) - h(q_i) + h(j) + \delta_{G_r}(u,q_j) \tag{23}$$
$$= \delta_{G_r}(v,q_i) + \delta_{G_r}(q_i,q_j) + \delta_{G_r}(u,q_j) \tag{24}$$
$$\leq (1+\varepsilon)(\delta_{G_r}(u,v)) \tag{25}$$
$$\leq (1+\varepsilon)(\delta_G(u,\lambda)) \tag{26}$$

Here, inequality (25) follows from (22), and (26) follows from that fact that P is fully contained in r, and our assumption that v is the closest λ-labeled vertex to u.

The proof for the case that $i \geq j$ is similar. □

5.1.1 Update

Assume that the label of u changes from λ_1 to λ_2. For every ancestor r of $r_u \in \mathcal{T}_G$, and $Q \in Sep_r$, and for $q_i \in C_{G_r}(u,Q)$, we remove from Pre_{Q,λ_1} and Suf_{Q,λ_1} the element (x,y) with $x = i$, and insert the element $(i, -h(i) + \delta_{G_r}(u,q_i))$ into Pre_{Q,λ_2}, and $(i, h(i) + \delta_{G_r}(u,q_i))$ into Suf_{Q,λ_2}. We note that since we assume that every vertex q_i is a portal of at most one vertex, the removals are well defined, and the insertions are safe.

The time and space bounds for the oracle described above are given in the following lemma.

Lemma 15 *Assume there exists a dynamic prefix/suffix minimum data structure in the word RAM model, that for a set of size m, supports PMQ/SMQ in $O(T_Q(m))$ time, and updates in $O(T_U(m))$ time, can be constructed in $O(T_C(m))$ time, where $T_C(m) \geq m$, and can be stored in $O(S(m))$ space. Then there exist a dynamic vertex-labeled stretch-$(1+\varepsilon)$ distance oracle for*

planar graphs with worst case query time $O(\varepsilon^{-1}\log(n)T_Q(\varepsilon^{-1}n))$, and expected amortized update time $O(\varepsilon^{-1}\log(n)T_U(\varepsilon^{-1}n))$. The oracle can be constructed using $O(\varepsilon^{-1}n\log^2 n + \log(n)T_C(\varepsilon^{-1}n))$ expected amortized time, and stored in $O(\log(n)S(\varepsilon^{-1}n))$ space.

Proof Let G be an undirected planar graph. We first decompose G to obtain $\mathcal{T}_G$, and compute all the portals and the distances to portals. Klein [11] shows that this can be done using $O(n\log(n)(\varepsilon^{-1}+\log n))$ time. Then, for every $r \in \mathcal{T}_G$, for every $Q \in Sep_r$ and every $\lambda \in \mathcal{L}_r$, we construct a prefix/suffix minimum query data structures for $Pre_{Q,\lambda}$ and $Suf_{Q,\lambda}$.

Recall that $Pre_{Q,\lambda}$ is defined over the set of pairs $\{(j, -h(q_j)+\delta_{G_r}(q_j,\lambda))\}_{j=0}^{k}$, where q_j is the j'th vertex on Q. For each element (x,y) in $Pre_{Q,\lambda}$, the first coordinate is specified by the order of the corresponding portal on Q. Hence, $x \le \varepsilon^{-1}n$ is an integer that fits in a single word. The second coordinate can be treated similarly; We sort the list $\{-h(q_j)+\delta_{G_r}(q_j,\lambda)\}_j$ for all portals q_j on Q. Then, we can specify y by its ordinal number in the sorted list. The same argument holds for $Suf_{Q,\lambda}$.

Constructing $Pre_{Q,\lambda}$ and $Suf_{Q,\lambda}$ takes $O(\log n(\varepsilon^{-1}n\log(\varepsilon^{-1}n)+T_C(\varepsilon^{-1}n))$ time, since at every level of $\mathcal{T}_G$ the total number of portals is $O(\varepsilon^{-1}n)$, and since $T_C(\cdot)$ is superlinear. The number of portals we store is $O(\varepsilon^{-1}n\log n)$ since every vertex v has $O(\varepsilon^{-1})$ portals for every one of its $O(\log n)$ ancestors in $\mathcal{T}_G$. Hence our space is $O(\log(n)S(\varepsilon^{-1}n))$, and the construction time is $O(\varepsilon^{-1}n\log^2 n + \log(n)T_c(\varepsilon^{-1}n))$.

To analyze the query and update time, we note that we process $O(\log n)$ nodes in $\mathcal{T}_G$ and in each we perform $O(\varepsilon^{-1})$ queries or updates to the prefix/suffix minimum query structures. The size of our prefix/suffix structures is bounded by the size of $V(Q)$ which is $O(\varepsilon^{-1}n)$. The ε^{-1} factor is due to the assumption of distinct portals. Thus, the query time is $O(\varepsilon^{-1}\log(n)T_Q(\varepsilon^{-1}n))$ and the update time is $O(\varepsilon^{-1}\log(n)T_U(\varepsilon^{-1}n))$.

Since every $Q \in Sep_r$ holds a prefix/suffix minimum data structure for every label $\lambda \in \mathcal{L}_r$, we use dynamic hashing to avoid space dependency on $|\mathcal{L}|$, as in Section 5. Hence, our construction time and update time are expected amortized. □

It remains to describe a fast prefix/suffix minimum query structure. We use a result due to Wilkinson [21] for solving the 2-sided reporting problem in $\mathbb{R}^2$ in the word RAM model. In this problem, we maintain a set A of n points in $\mathbb{R}^2$ under an online sequence of insertions, deletions and queries of the following form. Given a rectangle $B = [l_1,h_1]\times[l_2,h_2]$ such that exactly one of l_1, l_2 and one of h_1, h_2 is ∞ or $-\infty$, we report $A \cap B$. Here, $[l_1,h_1]\times[l_2,h_2]$ represents the rectangle $\{(x,y) : l_1 \le x \le l_2, h_1 \le y \le h_2\}$. Since Wilkinson assumes the word RAM model, it is assumed that the coordinates of the points in A are integers that fit in a single word. Wilkinson's data structure is captured by the following lemma.

Lemma 16 *[21, Theorem 5] For any $f \in [2, \log n/\log\log n]$, there exists a data structure for 2-sided reporting with update time $O((f\log n\log\log n)^{1/2})$,*

query time $O((f \log n \log\log n)^{1/2} + \log_f(n) + k)$ *where* k *is the number of points reported. The structure requires linear space.*

In fact, Wilkinson's structure first finds the point with the minimum y-coordinate in the query region, and then reports the other points. Using this fact, and setting $f = \log^{\gamma} n$ for some arbitrary small constant γ, we get the following lemma, in which we also state Wilkinson's construction time explicitly.

Lemma 17 *There exists a linear space data structure for 2-sided reporting on* n *points, with update time* $O(\log^{1/2+\gamma} n)$ *and query time* $O(\frac{\log n}{\log\log n})$*. This data structure can be constructed in* $O(n \log^{1/2+\gamma} n)$ *time. Moreover, upon query the data structure returns the minimum* y*-coordinate of a point in the query region.*

The prefix/suffix queries required by Lemma 15 correspond to one-sided range reporting in the plane, which can be solved using 2-sided queries, by setting the upper limit of the query rectangle to nN.

Lemma 18 *For any constant* $\gamma > 0$*, there exists a linear space dynamic prefix/suffix minimum data structure over* n *elements with update time* $O(\log^{1/2+\gamma} n)$*, and query time* $O(\frac{logn}{\log\log n})$*. This data structure can be constructed in* $O(n \log^{1/2+\gamma} n)$ *time.*

Proof We use Wilkinson's structure. A prefix minimum query for i corresponds to finding the point with minimum y-coordinate in the rectangle $(-\infty, -\infty, i, \infty)$. This is a 1-sided rectangle. To be able to specify a boundry for the y-axis, we maintain an upper bound y_{max} on the y-coordinates of points in A. The bound can be easily updated in constant time when an insertion occurs. (There is no need to update the bound when a deletion occurs). We replace the 1-sided rectangle with the 2-sided rectangle $(-\infty, -\infty, i, y_{max})$. Similarly, our suffix minimum query is the 1-sided rectangle $(i, -\infty, \infty, \infty)$ or the 2-sided $(i, -\infty, \infty, y_{max})$. The lemma now follows by applying Lemma 17. □

Plugging Lemma 18 into Lemma 15 proves Theorem 3.

6 Conclusion

In this paper we presented approximate vertex-labeled distance oracles for directed and undirected planar graphs with polylogarithmic query and update times and nearly linear space. All of our oracles have $\Omega(\log n)$ query and updates since we handle root-to-leaf paths in the decomposition tree. It would be interesting to study whether this can be avoided, as done in the vertex-to-vertex case, where approximate distance oracles with faster query times exist (see e.g., [5,19,24] and references therein). Another interesting question that arises is that of faster dynamic prefix minimum data structures. In Section 5 we used Wilkinson's 2-sided reporting [21] as a dynamic prefix/suffix minimum data structure. Can other approaches to this problem be used to obtain a faster solution?

Acknowledgements We thank Paweł Gawrychowski and Oren Weimann for fruitful discussions.

References

1. Abraham, I., Chechik, S., Delling, D., Goldberg, A.V., Werneck, R.F.: On dynamic approximate shortest paths for planar graphs with worst-case costs. In: SODA, pp. 740–753. SIAM (2016)
2. Abraham, I., Chechik, S., Gavoille, C.: Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In: STOC, pp. 1199–1218. ACM (2012)
3. Chechik, S.: Improved distance oracles and spanners for vertex-labeled graphs. In: ESA, *Lecture Notes in Computer Science*, vol. 7501, pp. 325–336. Springer (2012)
4. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM **31**(3), 538–544 (1984). DOI 10.1145/828.1884. URL `http://doi.acm.org/10.1145/828.1884`
5. Gu, Q., Xu, G.: Constant query time $(1 + \epsilon)$-approximate distance oracle for planar graphs. In: ISAAC, pp. 625–636 (2015)
6. Henzinger, M.R., Klein, P.N., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. Syst. Sci. **55**(1), 3–23 (1997). DOI 10.1006/jcss.1997.1493. URL `https://doi.org/10.1006/jcss.1997.1493`
7. Hermelin, D., Levy, A., Weimann, O., Yuster, R.: Distance oracles for vertex-labeled graphs. In: ICALP (2), *Lecture Notes in Computer Science*, vol. 6756, pp. 490–501. Springer (2011)
8. Kawarabayashi, K., Klein, P.N., Sommer, C.: Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In: ICALP (1), *Lecture Notes in Computer Science*, vol. 6755, pp. 135–146. Springer (2011)
9. Kawarabayashi, K., Sommer, C., Thorup, M.: More compact oracles for approximate distances in undirected planar graphs. In: SODA, pp. 550–563. SIAM (2013)
10. Klein, P.N.: Preprocessing an undirected planar network to enable fast approximate distance queries. In: SODA, pp. 820–827 (2002)
11. Klein, P.N.: Multiple-source shortest paths in planar graphs. In: SODA, pp. 146–155 (2005)
12. Li, M., Ma, C.C.C., Ning, L.: $(1 + \epsilon)$-distance oracles for vertex-labeled planar graphs. In: TAMC, pp. 42–51 (2013)
13. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics **36**(2), 177–189 (1979)
14. Łącki, J., Ocwieja, J., Pilipczuk, M., Sankowski, P., Zych, A.: The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In: STOC, pp. 11–20 (2015)
15. Mozes, S., Skop, E.E.: Efficient vertex-label distance oracles for planar graphs. In: WAOA, pp. 97–109 (2015)
16. Mozes, S., Skop, E.E.: Efficient vertex-label distance oracles for planar graphs. Theory of Computing Systems (2017). DOI 10.1007/s00224-017-9827-0. URL `https://doi.org/10.1007/s00224-017-9827-0`
17. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: ESA, pp. 121–133 (2001)
18. Sommer, C.: Shortest-path queries in static networks. ACM Comput. Surv. **46**(4), 45:1–45:31 (2014). DOI 10.1145/2530531. URL `http://doi.acm.org/10.1145/2530531`
19. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**(6), 993–1024 (2004). DOI 10.1145/1039488.1039493. URL `http://doi.acm.org/10.1145/1039488.1039493`
20. Thorup, M., Zwick, U.: Approximate distance oracles. In: STOC, pp. 183–192. ACM (2001)
21. Wilkinson, B.T.: Amortized bounds for dynamic orthogonal range reporting. In: ESA, pp. 842–856 (2014)
22. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space theta(n). Inf. Process. Lett. **17**(2), 81–84 (1983). DOI 10.1016/0020-0190(83)90075-3. URL `http://dx.doi.org/10.1016/0020-0190(83)90075-3`

23. Wulff-Nilsen, C.: Approximate distance oracles with improved preprocessing time. In: SODA, pp. 202–208. SIAM (2012)
24. Wulff-Nilsen, C.: Approximate distance oracles for planar graphs with improved query time-space tradeoff. In: SODA, pp. 351–362 (2016)