The Interdisciplinary Center, Herzlia
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Efficient Dynamic Approximate Distance Oracles for Vertex-Labeled Planar Graphs

Submitted by Itay Laish
Under the supervision of Dr. Shay Mozes.

M.Sc. dissertation, submitted in partial fulfillment of the requirements
for the M.Sc. degree, research track, School of Computer Science
The Interdisciplinary Center, Herzliya

July 2017

Acknowledgements

First of all, I would like to thank my advisor Dr. Shay Mozes, for providing me much needed directions, especially, when I was sure I'm facing a dead end, and for introducing me to the world of research, and investing exponential amount of time, teaching and tutoring me. His way of thinking and passion to find efficient, yet elegant solutions inspired me, and gave me the drive to look deeper and wider, and to seek for ideas in different places.

Second, I would like to thank Paweł Gawrychowski and Oren Weimann for fruitful discussions, and for well appreciated tips, that found their way into this research.

Abstract

Let G be a graph where each vertex is associated with a label. An **Approximate Vertex-Labeled 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 thesis 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 previously known approximate vertex-labeled distance oracle supported both queries and updates in sublinear time.

Contents

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 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 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 fixed parameter $\varepsilon \geq 0$. 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

Approximate vertex-to-vertex distance oracles

We outline related results, and refer the reader to an extensive survey due to Sommer [Som14]. For general graphs, Thorup and Zwick [TZ01] 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 [Wul12] 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 [Tho04] 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 [Kle02, Kle05] independently described a stretch-$(1+\varepsilon)$ distance oracle for undirected graphs with the same bounds, but faster preprocessing time. Kawarabayashi, Klein and Sommer [KKS11], 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 [KST13] 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 [ACG12] presented a stretch-$(1+\varepsilon)$ distance oracle that supports both query and edge length updates in $\tilde{O}(n^{1/2})$ time worst case. Abraham et al. [ACD+16] 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 label change.

Approximate vertex-to-label distance oracles

The vertex-to-label query problem was introduced at ICALP'11 by Hermelin, Levy, Weimann and Yuster [HLWY11]. 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, $(2 \cdot 3^{k-1} + 1)$. Chechik [Che12] 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, Ma and Ning [LMN13]. They described a stretch-$(1+\varepsilon)$ distance oracle that is based on Klein's results [Kle02]. 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 [MS15], 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 Ma and Ning [LMN13] considered the dynamic case, but their update method was trivial and takes $\Theta(n \log n)$ time in the worst case. Łącki et al. [aOP+15] presented a different dynamic vertex-to-label oracle for undirected planar graphs, in the context of computing Steiner trees. Their orcale 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 (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.

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 [Tho04] is required.

Our second oracle for undirected graphs (Section 5) uses the approach taken by Li, Ma and Ning [LMN13] 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 [Wil14].

Our oracle for directed planar graphs (Section 4) is based on the static vertex-labeled distance oracle of [MS15], 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. The following table summarizes the comparison between our oracles and the relevant previously known ones.

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

Table 1.1: Vertex-to-Label Distance Oracles Time Bound Comparison

	D/U	Query time	Update time
Li, Ma and Ning [LMN13]	U	$O(\varepsilon^{-1} \log n \log \Delta)$	$O(n \log n)$
Łącki at el. [aOP+15]	U	$O(\varepsilon^{-1} \sqrt{n} \log^2 n \log D)$	$O(\varepsilon^{-1} \sqrt{n} \log^2 n \log D)$
Section 3 (faster query)	U	$O(\varepsilon^{-1} \log n \log \log nN)$	$O(\varepsilon^{-1} \log n \log \log \varepsilon^{-1} \log nN)$
Section 5 (faster 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 and Skop [MS15]	D	$O(\varepsilon^{-1} + \log \log n \log \log nN)$	N/A
Section 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

An undirected (respectively directed) graph G is a tuple consisting of a finite set of objects named vertices, denoted by $V(G)$, and a set of edges (resp, arcs) denoted by $E(G)$. An undirected (resp. directed) edge (resp. arc), is an unordered (resp. ordered) pair of vertices. Given an edge $e = uv$, we say that u and v are the *endpoints* of e. If e is an arc (in a directed graph), we say that the orientation of e is from u to v. An undirected (resp. directed) *u-to-v path* P is a sequence of vertices in $V(G)$, that starts with u, ends with v, and for every pair $(x, y) \in P$ there exists an edge (resp. an arc) $xy \in E(G)$. A path P is called a *simple path* if every vertex in $V(G)$ appears at most once in P. A *cycle* C is a path that begins and ends in the same vertex. Analogously, a *simple cycle* is a cycle such that every vertex in C appears at most once, except from the first vertex, that is also the last.

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 the graph G is *connected* if for every $u, v \in V(G)$ there exists an u-to-v path in G'.

An *undirected tree* T is an undirected connected graph that contains no cycles. I.e. for every pair of vertices u, v there exists a unique path from u to v. A vertex $v \in V(T)$ is called a *leaf* of T if it is adjacent to at most one other vertex. A *rooted undirected tree* is a tree T equipped with a vertex $r \in V(T)$ referred to as root. A *spanning tree* T of a graph G is a subgraph of G (respectively, if G is directed, T is a subgraph of the underlying undirected graph G') consisting of $|V(G) - 1|$ edges from $E(G)$ (resp. $E(G')$), that forms a tree, such that for every $v \in V(G)$ there exists a r-to-v path in T.

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 closest vertex to the root of T on the v-to-*root* path that fulfills A.

Given an undirected graph G with a spanning tree T rooted at r and an edge uv not in T, the *fundamental cycle* of uv (with respect to T) is the 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 $\ell(\cdot)$. Let P be a path from u-to-v. The length of P is $\Sigma_{e \in P} \ell(e)$. The shortest u-to-v path is a path that minimizes that distance. We define the distance form 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 *the shortest path* between two vertices. Our data structures do not require this assumption. 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 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 $\varepsilon \geq 0$, 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 $\alpha, \varepsilon \geq 0$, a* scale-$(\alpha, \epsilon)$ 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 ∞.*

The only properties of planar graphs that we use in this paper are the existence of shortest path separators (see below), and the fact that single source shortest paths can be computed in $O(n)$ time in a planar graph with n vertices [HKRS97].

Definition 3. *Let G be a directed graph. Let G' be the undirected graph induced by G. Let P be a 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 (the undirected path corresponding to) each shortest path in S is a subpath of P and each vertex of P is in some shortest path in S.*

Definition 4. *Let G be a directed embedded planar graph. An undirected cycle C is a* balanced cycle separator *of G if each of the strict interior and the strict exterior of C contains at most $2|V(G)|/3$ vertices. 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 [LT79, Tho04] that for any spanning tree of G, there exists a fundamental cycle C 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.

2.1 Existing Techniques for Approximate Distance Oracles for Planar Graphs

Thorup shows that to obtain a stretch-$(1+\varepsilon)$ distance oracle, it suffices to show scale-(α, ε) oracles for so-called α-*layered* graphs. An α-layered graph is one equipped with a 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. *[Tho04, Lemma 3.9] For any planar graph G and fixed parameter ε, a stretch-$(1+\varepsilon)$ distance oracle can be constructed using $O(\log nN)$ scale-(α, ε') distance oracles for α-layered graphs, where $\alpha = 2^i$, $i = 0, ...\lceil \log nN \rceil$ and $\varepsilon' \in \{1/2, \varepsilon/4\}$. If the scale-$(\alpha, \varepsilon')$ has query time $t(\epsilon')$ independent of α, the stretch-$(1+\varepsilon)$ distance oracle can answer queries in $O(t(1/2)\varepsilon^{-1} + t(\varepsilon/4) \log\log(nN))$.*

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

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

We recursively decompose 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 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. Note that C_r belongs to both G_{r_1} and G_{r_2}. For a vertex $v \in V(G)$, we denote by r_v the leaf node of $\mathcal{T}_G$ that contains v. See Fig. 2.1 for an illustration.

We now describe the basic building block used in our (and in many previous) distance oracle. 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, is not efficient since the number of vertices on Q might be $\theta(|V(G)|)$. 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)**.** *[Tho04, Section 3.2.1] Let $\varepsilon, \alpha \geq 0$ 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 $\delta_G(u,q) + \delta_G(q,t) \leq \delta_G(u,t) + \varepsilon\alpha$.*

One defines (α,ε)-covering connections sets $C_G(Q,u)$ from Q to u symmetrically. Thorup proves that there always exists an (α,ε)-covering connections set of size $O(\varepsilon^{-1})$:

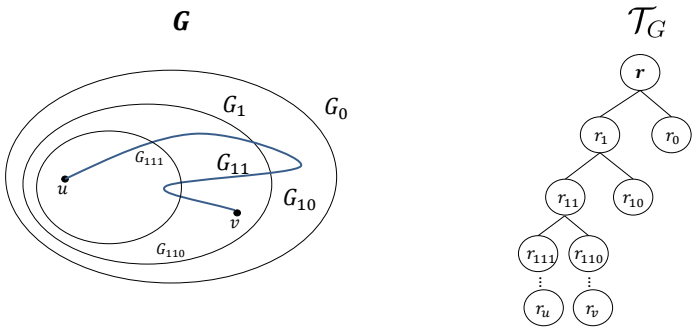


Figure 2.1: An illustration of (part of) the recursive decomposition of a graph G using cycle separators, and the corresponding decomposition tree $\mathcal{T}_G$. The graph G is decomposed using a cycle separator into G_0, and G_1. Similarly, G_1 is decomposed into G_{10} and G_{11}, and G_{11} is decomposed 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 $G_{r_1} = G_1$.

Lemma 2. *[Tho04, 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.*

We will use the term ε-covering connections set whenever α is obvious from the context. Thorup shows that (α, ε)-covering connections sets can be computed efficiently.

Lemma 3. *[Tho04, 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* $Q \in Sep_r$*.*

3 An Oracle for Undirected Graphs With Faster Query

Let H be an undirected α-layered graph,[1] and let T be the associated spanning tree of H. For any fixed parameter ε' we set $\varepsilon = \frac{\varepsilon'}{3}$. We decompose H using shortest path separators w.r.t. T. Let $\mathcal{T}_H$ be the resulting decomposition tree. 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 along Q.[2] Thus, for every vertex $t \in V(Q)$ there is a vertex $q \in C_Q$ such that $\delta_H(t,q) \leq \varepsilon\alpha$.

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 [HKRS97]. This computes the connection lengths $\delta_{H_r}(u,q)$, for all $u \in V(H_r)$.

Lemma 4. *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 2ε-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 $\delta_{H_r}(u,t) \leq \delta_{H_r}(u,q) + \delta_{H_r}(q,t) \leq \delta_{H_r}(u,t) + \varepsilon'\alpha$. Since $t \in Q$, there exists a vertex $q \in C_Q$ such that $\delta_H(q,t) \leq \varepsilon\alpha$. Since H is undirected, the triangle inequality for shortest path lengths holds for any three vertices in $V(H)$.

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

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

We start with the triangle inequality between u, t and q in H as follows.

$$\delta_{H_r}(u,q) \leq \delta_{H_r}(u,t) + \delta_{H_r}(t,q) \tag{3.1}$$
$$\delta_{H_r}(u,q) + \delta_{H_r}(t,q) \leq \delta_{H_r}(u,t) + \delta_{H_r}(t,q) + \delta_{H_r}(t,q) \tag{3.2}$$
$$\delta_{H_r}(u,q) + \delta_{H_r}(t,q) \leq \delta_{H_r}(u,t) + 2\varepsilon\alpha \tag{3.3}$$

From the triangle inequality, $\delta_{H_r}(u,t) \leq \delta_{H_r}(u,q) + \delta_{H_r}(q,t)$, and the lemma follows. □

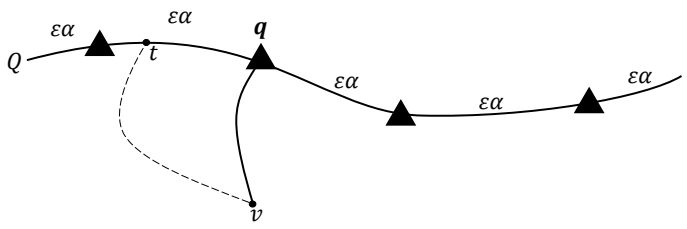


Figure 3.1: Illustration of Lemma 4. Q is a shortest path in some separator, the connections of C_Q are marked by triangles. The solid v-to-q path reflects the shortest path from v to the connection q, and the dashed v-to-t path reflects the shortest path from v to t.

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 value such that $\delta_{H_r}(q,v) \leq k\varepsilon\alpha$. Thus, $\delta_{H_r}(q,v) \leq \hat{\delta}_{H_r}(q,v) \leq \delta_{H_r}(q,v) + \varepsilon\alpha$. Let $L_r(q,\lambda)$ be the list of the distances $\hat{\delta}_{H_r}(q,v)$ for all $v \in S_r^{\lambda}$. 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. 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.

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 also inspect the λ-labeled vertices in H_{r_u} explicitly. We return the minimum distance inspected. See Fig. 3.2 for an illustration.

Lemma 5. *The query algorithm runs in $O(\varepsilon^{-1}\log n)$ time, and returns a distance d such that $\delta_H(u,\lambda) \leq d \leq \delta_H(u,\lambda) + 3\varepsilon\alpha$.*

Proof. Let v be the closest λ-labeled to u in H. It is trivial that if the shortest path P form u-to-v does not leave $r_u = r_v$ the query algorithm is correct, since the distances in r_u are computed 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 it is also the closest λ-labeled vertex to t.

Since $t \in Q$, there exists $q \in C_Q$ such that $\delta_{H_r}(v,q) + \delta_{H_r}(q,t) \leq \delta_{H_r}(v,t) + \varepsilon'\alpha$. By the triangle, $\delta_{H_r}(v,q) \leq \delta_{H_r}(q,t) + \delta_{H_r}(v,t)$. Hence, $first(L_r(q,\lambda)) \leq \delta_{H_r}(q,t) + \delta_{H_r}(v,t) \leq \delta_{H_r}(q,v) + \varepsilon\alpha$.

$$\begin{aligned} first(L_r(q,\lambda) &\leq \hat{\delta}_{H_r}(q,v) \leq \delta_{H_r}(q,v) + \varepsilon\alpha && (3.4)\\ &\leq \delta_{H_r}(q,t) + \delta_{H_r}(t,v) + \varepsilon\alpha && (3.5)\\ &\leq \delta_{H_r}(t,v) + 2\varepsilon\alpha && (3.6) \end{aligned}$$

Where inequality (3.4) follows from the definition of $L_r(q,\lambda)$, (3.5) follows from the triangle inequality, and (3.6) follows from the fact that $\delta_{H_r}(q,t) \leq$

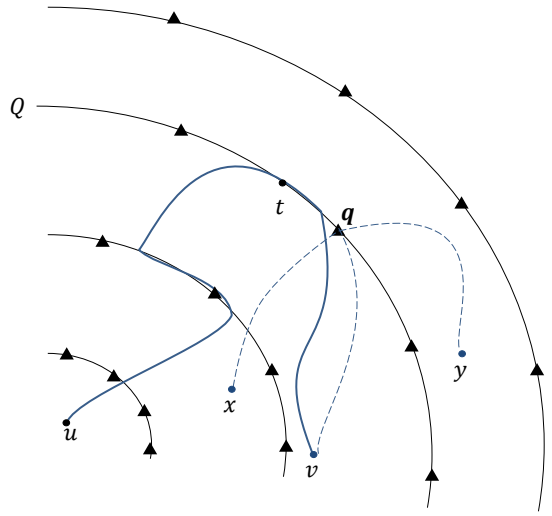


Figure 3.2: Illustration of the query algorithm. The solid quarter-circles are shortest paths of separators in G. The vertices x, y and v have label λ, and v is the closest λ-labeled vertex to u. The path Q belongs to the root-most node r whose separator is intersected by the shortest u-to-λ path (solid blue). The vertices q and t on Q are as in the proof of Lemma 5. 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.

$\varepsilon\alpha$.

$$\begin{aligned}
Query(u,\lambda) \leq & \delta_{H_r}(u,q) + first(L_r(q,\lambda)) && (3.7)\\
\leq & \delta_{H_r}(u,q) + \delta_{H_r}(t,v) + 2\varepsilon\alpha && (3.8)\\
\leq & \delta_{H_r}(u,t) + \delta_{H_r}(t,q) + \delta_{H_r}(t,v) + 2\varepsilon\alpha && (3.9)\\
\leq & \delta_{H_r}(u,v) + 3\varepsilon\alpha && (3.10)\\
\leq & \delta_H(u,\lambda) + 3\varepsilon\alpha && (3.11)
\end{aligned}$$

Here, inequality (3.9) follows from the triangle ineqaulity, and (3.11) follows from the fact that P is fully contained in H_r, and our assumption that v is the closest λ-labeled vertex to v.

Since $\delta_{H_r}(u,q) + first(L_r(q,\lambda))$ underlines a real path in the H_r, from our assumption that v is the closest λ-labeled vertex to u, it follows that $Query(u,\lambda) \geq \delta_{H_r}(u,v)$, and the lemma follows.

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 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. For $r \in \mathcal{T}_H$, let $\mathcal{L}_r$ be 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 5 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 [FKS84] with expected construction time and constant query time. In the dynamic case, we will use a dynamic hashing scheme, e.g., [PR01], 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.

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 $ID_q(v)$. 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 list as before, we implement $L_r(q,\lambda)$ using a successor/predecessor structure over the universe $[1,...,|V(H_r)|]$ of the IDs. For example, we can use y-fast tries [Wil83] that support operations in $O(\log\log\varepsilon^{-1})$ expected amortized time and minimum query in $O(1)$ worst case.

$Query(u,\lambda)$

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 $L_r(q,\lambda)$, and use the lookup table to get the actual distance between q and the vertex with that ID.

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 $ID_q(v)$ from $L_r(q,\lambda_1)$ and insert it to $L_r(q,\lambda_2)$.

Lemma 6. *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. 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 node, and is bounded by $O(\log n)$ expected amortized time in total. □

Lemma 7. *The data structure can be constructed in $O(\varepsilon^{-1}n\log n\cdot\log\log\varepsilon^{-1})$ expected amortized time, and stored using $O(\varepsilon^{-1}n\log n)$ space.*

Proof. We decompose H into $\mathcal{T}_H$, and compute the connection length in $O(\varepsilon^{-1}n\log n)$ time. We than build the lists $L_r(q)$ for every node $r \in \mathcal{T}_H$

and q on any $q \in Sep_r$. These lists contains $O(\varepsilon^{-1} n \log n)$ 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 than 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 $O(\varepsilon^{-1} n \lg n \log\log \varepsilon^{-1})$ expected amortized. To see our space bound, we note that every v contributes a distance $O(\varepsilon^{-1})$ 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 has linear space in the number of elements stored. Thus, $O(\varepsilon^{-1} n \log n)$ space. □

We plug in this structure to Lemma 1 and obtain the following theorem:[3]

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 and updates in $O(\varepsilon^{-1} \log n \cdot \log\log \varepsilon^{-1} \log nN)$ expected amortized. 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.*

[3]Formally, one needs to show that Lemma 1 holds for vertex-labeled oracles as well. See Appendix A

4 Oracle for Directed Graphs

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. To describe our data structure for directed graphs, we first need to introduce the concept of 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 $\delta(S,q)$ (compare to Definition 5) 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 planar 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 2 on x w.r.t 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 the vertices in $\bigcup_{i=1}^{k} D_Q(S_i)$ and q_0. Let $\hat{H}$ be the auxiliary graph consisting of $\hat{Q}$ and an apex vertex x connected to every $q \in \hat{Q}$ with an arc xq of length $\delta_H(S_i,q)$, where S_i is the set originally containing q. Note that $\delta_H(S_i,q) \geq \delta_H(S,q)$. Also note that $\hat{H}$ is planar, with 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 relaxing all arcs adjacent to x, and than relaxing the arcs of Q by order. Constructing $\hat{H}$ and computing these distances can be done in $O(m)$ time, since $|V(\hat{H})| = |E(\hat{H})| = O(m)$.

We apply Lemma 2 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 $\hat{Q}$. It remains to prove that $C_{\hat{H}}(x,Q)$ is an $(\varepsilon+\varepsilon')$-covering set $C_H(S,Q)$ 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 $D_Q(S_1)$ is an ε'-covering set from S_1 to Q in H, there exists $(q',\ell') \in D_Q(S_1)$ such that:

$$\ell' + \delta_H(q',t) \leq \delta_H(S_1,t) + \varepsilon'\alpha \tag{4.1}$$

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

$$\ell + \delta_{\hat{H}}(q,q') \leq \delta_{\hat{H}}(x,q') + \varepsilon\alpha \leq \ell' + \varepsilon\alpha \tag{4.2}$$

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

$$\begin{aligned}
\delta_H(S_1 \cup S_2,t) + \varepsilon'\alpha + \varepsilon\alpha &= \delta_H(S_1,t) + \varepsilon'\alpha + \varepsilon\alpha && (4.3)\\
&\geq \ell' + \delta_H(q',t) + \varepsilon\alpha && (4.4)\\
&\geq \ell + \delta_{\hat{H}}(q,q') + \delta_H(q',t) && (4.5)\\
&= \ell + \delta_H(q,q') + \delta_H(q',t) && (4.6)\\
&= \ell + \delta_H(q,t) && (4.7)
\end{aligned}$$

Here, (4.4) follows from inequality (4.1), (4.5) follows from inequality (4.2). □

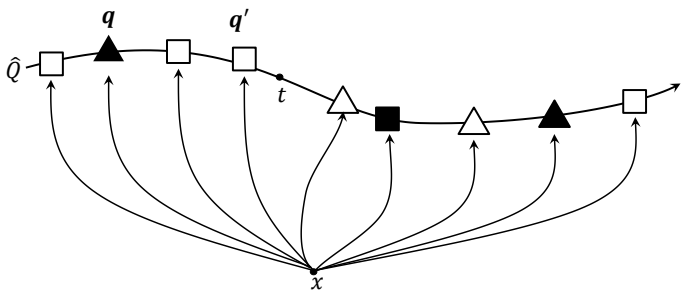
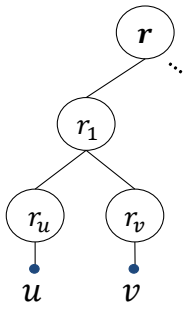


Figure 4.1: Illustration of he auxiliary graph $\hat{H}$ in the proof of Lemma 9, when applied to the sets $S_1 = u$ and $S_2 = v$ (I.e. $k = 2$). The connection sets of S_1 and S_2 are indicated by the triangles and squares, respectively. The connections in the output set are indicated by a solid fill. 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.

Let H be a directed planar α-layered graph, equipped with a spanning tree T. For every fixed parameter ε, let $\hat{\epsilon} = \frac{\varepsilon}{8 \log n}$, and $\varepsilon^* = \frac{\varepsilon}{2}$. We apply Lemma 3 with $\hat{\epsilon}$ to H and obtain a decomposition tree $\mathcal{T}_H$, and $\hat{\epsilon}$-covering sets $C_{H_r}(v, Q)$ and $C_{H_r}(Q, v)$ for every $v \in V(H)$, every ancestor r of r_v in $\mathcal{T}_H$ and every $Q \in Sep_r$. For every $1 \leq i \leq \log n$, let $\varepsilon_i = \frac{\varepsilon \log n - i + 1}{4 \log n}$.

For every $r \in \mathcal{T}_H$, for every $\lambda \in \mathcal{L}_r$, and for every $Q \in Sep_r$, we apply Lemma 9 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 $\frac{\varepsilon}{4}$. Thus, we obtain an ε^*-covering set $C^*_{H_r}(S_r^\lambda, Q)$. Let i be the level of r in $\mathcal{T}_H$, we also store for r a set of ε_i-covering sets as follows. For every ancestor node t of r in $\mathcal{T}_H$ and every $Q \in Sep_t$, we store $C_{H_t}(S_r^\lambda, Q)$. We assume for the moment that these sets are given. We defer



	Vertex connections $\hat{\varepsilon}$-covering sets	Query connections ε^*-covering sets	Update connections ε_i-covering sets
r	N/A	From S_G^λ to $Q \in Sep_r$	From S_G^λ to $Q \in Sep_r$
r_1	N/A	From $S_{r_1}^\lambda = S_{r_u} \cup S_{r_v}$ to $Q \in Sep_{r_1}$	From $S_{r_1}^\lambda = S_{r_u} \cup S_{r_v}$ to $Q \in \{Sep_{r_1}, Sep_r\}$
r_u	N/A	From $S_{r_u}^\lambda = u$ to $Q \in Sep_{r_u}$	From $S_{r_u}^\lambda = u$ to $Q \in \{Sep_{r_u}, Sep_{r_1}, Sep_r\}$
r_v	N/A	From $S_{r_v}^\lambda = v$ to $Q \in Sep_{r_v}$	From $S_{r_v}^\lambda = v$ to $Q \in \{Sep_{r_v}, Sep_{r_1}, Sep_r\}$
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

Figure 4.2: A summary of the connections-sets stored by the directed oracle. On the left, part of a decomposition tree of a graph is shown. The vertices u and v are the only λ labeled vertices. On the right, a table listing all the covering sets that are stored for the label λ.

the description of their construction (see the update procedure and the proof of Lemma 12). We will use the ε^*-covering sets for efficient queries, and the more accurate ε_i-covering sets to be able to perform efficient updates. See Fig. 4.2.

4.1 $Query(\lambda, u)$

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

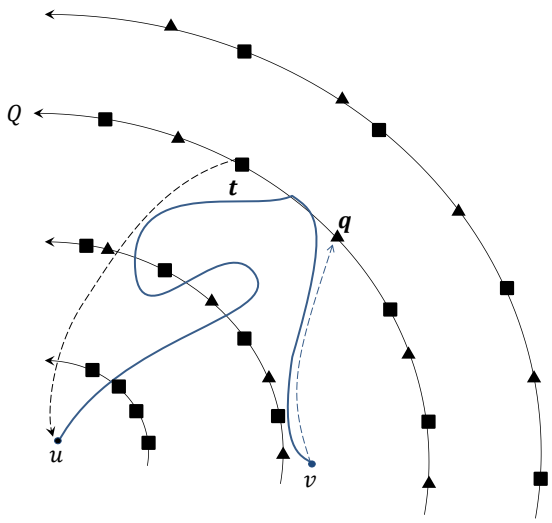


Figure 4.3: The solid quarter-circles are shortest paths of separators in G. The vertex v is the closest λ-labeled vertex to u. The path Q belongs to the root-most node r whose separator is intersected by the shortest λ-to-u path (solid blue). The connections of S_r^λ on Q are indicated by black triangles, the connections of u are indicated by black squares. The vertices q and t on Q are as in the proof of Lemma 10. 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.

Lemma 10. $Query(\lambda, u) \leq \delta_H(\lambda, u) + \varepsilon\alpha$.

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 connection set $C^*_{H_r}(S_r^\lambda, Q)$, there exists (q, ℓ) such that

$$l + \delta_{H_r}(q, k) \leq \delta_{H_r}(S_r^\lambda, k) + \varepsilon^*\alpha \tag{4.8}$$

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

$$\delta_{H_r}(k, t) + \delta_{H_r}(t, u) \leq \delta_{H_r}(k, u) + \hat{\epsilon}\alpha \leq \delta_{H_r}(k, u) + \varepsilon^*\alpha \tag{4.9}$$

We add the two to get

$$l + \delta_{H_r}(q,k)\delta_H(k,t) + \delta_{H_r}(t,u) \leq \delta_{H_r}(S_r^\lambda, k) + \delta_{H_r}(k,u) + \varepsilon^*\alpha + \varepsilon^*\alpha \tag{4.10}$$

$$l + \delta_{H_r}(q,t) + \delta_H(t,u) \leq \delta_{H_r}(S_r^\lambda, k) + \delta_{H_r}(k,u) + 2\varepsilon^*\alpha \tag{4.11}$$

$$l + \delta_{H_r}(q,t) + \delta_{H_r}(t,u) \leq \delta_{H_r}(S_r^\lambda, u) + \varepsilon\alpha \tag{4.12}$$

Clearly, $l + \delta_H(q,t) + \delta_H(t,u) \geq \delta_H(S_r^\lambda, u)$. And 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 $S_{r_u}^{\lambda_1}$. We stress that for every $1 \leq j \leq k$, v_j has an $\hat{\epsilon}$-covering set $C_{H_t}(v_j, Q)$ of size $O(\hat{\epsilon}^{-1})$ 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 $\varepsilon' = \hat{\epsilon}$ and ε set to $\hat{\epsilon}$. Lemma 9 yields a $2\hat{\epsilon}$-covering set $C_{H_t}(S_{r_u}^\lambda, Q)$.

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, $S_r^{\lambda_1} = S_x^{\lambda_1} \cup S_y^{\lambda_1}$. Therefore, by Lemma 9, for every ancestor t of r, and every $Q \in Sep_t$, $C_{H_t}(S_r^{\lambda_1}, Q)$ can be obtained from $C_{H_t}(S_x^{\lambda_1}, Q) \cup C_{H_t}(S_y^{\lambda_1}, Q)$. Let i by 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 (y) stores an ε_{i+1}-covering set $C_{H_t}(S_x^{\lambda_1}, Q)$ ($C_{H_t}(S_y^{\lambda_1}, Q)$). We apply Lemma 9 on $C_{H_t}(S_x^{\lambda_1}, Q)$ and $C_{H_t}(S_y^{\lambda_1}, Q)$ with $\varepsilon' = \varepsilon_{i+1}$ and $\varepsilon = 2\hat{\epsilon}$ to get an $(\varepsilon_{i+1} + 2\hat{\epsilon})$-covering set $C_{H_t}(S_r^{\lambda_1}, Q)$. The following lemma shows that $C_{H_t}(S_r^{\lambda_1}, Q)$ is an ε_i-covering set.

Lemma 11. *Let r be a node in level i in $\mathcal{T}_H$. For every ancestor t of r, and every $Q \in Sep_t$, $C_{H_t}(S_r^{\lambda_1}, Q)$ is an ε_i-covering set from $S_r^{\lambda_1}$ 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 connection sets of the leaf nodes are computed explicitly using Lemma 9, with ε and ε' set to $\hat{\epsilon}$. Hence the product of the lemma is $2\hat{\epsilon}$-covering sets.

$$2\hat{\epsilon} = 2\frac{\varepsilon}{8 \log n} = \frac{\varepsilon}{4 \log n} = \varepsilon_{\log n} \tag{4.13}$$

For the inductive step, if r is a leaf, then the arguments from the base case applies. 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 $S_x^{\lambda_1}$ and $S_y^{\lambda_1}$ to Q, respectively. The update procedure applies Lemma 9 on $C_{H_x}(S_x^{\lambda_1}, Q)$ and $C_{H_y}(S_y^{\lambda_1}, Q)$ with $\varepsilon' = \varepsilon_{i+1}$ and $\varepsilon = 2\hat{\epsilon}$, so we get an $(\varepsilon_{i+1} + 2\hat{\epsilon})$-covering set $C_{H_t}(S_r^{\lambda_1}, Q)$.

$$\varepsilon_{i+1} + 2\hat{\epsilon} = \varepsilon\frac{\log n - (i+1) + 1}{4 \log n} + \varepsilon\frac{2}{8 \log n} = \varepsilon\frac{\log n - i + 1}{4 \log n} = \varepsilon_i \tag{4.14}$$

□

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 in level i on $\mathcal{T}_H$. By Lemma 11, for every $Q \in Sep_r$, we have an ε_i-covering set $C_{H_r}(S_r^{\lambda_1}, Q)$. Since $\varepsilon_i < \varepsilon^*$, $C_{H_r}(S_r^{\lambda_1}, Q)$ is also an ε^*-covering set. However, it is too large. We apply Lemma 9 on $C_{H_r}(S_r^{\lambda_1}, Q)$ 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 $C^*_{H_r}(S_r^{\lambda_1}, Q)$. We repeat the entire process for λ_2.

Lemma 12. *There exists a scale-(α, ε) distance oracle for directed α-layered planar graph, with query time $O(\varepsilon^{-1} \log n)$ worst case, and update time of $O(\varepsilon^{-1} \log^3 n)$ expected amortized. 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. 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 sizes of the input connection sets we get the bound

by the number of connection stored for r_u and its ancestors. We store for r_u $\frac{\varepsilon}{4\log n}$-connection set for constant number of separators for every one of the $O(\log n)$ ancestors of r_u. Hence, the number of connections stores 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 connection stored for any other strict ancestor of r_u, we get the total number of connections stored of $O(\log^3 n)$.

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 connection of its children. Thus, any connection is used at most twice. Once for updating a connection set of its parent, and the second time, is when updating the ε^*-covering sets of r (r_u). Hence the total input size of Lemma 9 is at most twice the number of the connections stored for the ancestors of r_u, that is $O(\log^3 n)$, and the update time follows.

Since we store for every $r \in \mathcal{T}_H$ and every $Q \in Sep_r$ a connection 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 $O(\varepsilon^{-1}\log n)$ time worst case.

By Lemma 3 with ε set to $\hat{\epsilon}$, all connection sets for all leaves of $\mathcal{T}_H$ can be computed in $O(\varepsilon^{-2}n\log^5 n)$ and it requires $O(\varepsilon^{-1}n\log^2 n)$ space. We construct the connection 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 operation, and $O(\varepsilon^{-1}n\log^3 n)$ expected amortized time in total. This is dominated by the construction of Thorup's oracle.

To get the space requirements of our data structure, we need to count the number of connections stored. If every vertex $u \in V(H)$ has unique label, it follows that the connection sets stored for u are not useful for any other vertex. We therefore count the number of u's connections and multiple by $O(n)$. Let λ be the label of u. To support queries and updates for λ, we store for every ancestor r of r_u connection sets from S_r^λ to $O(\log n)$ separators for the ancestors of r. Since the size of these connection sets is only bounded by $O(\hat{\varepsilon}^{-1})$, we get that r requires $O(\varepsilon^{-1}\log^2 n)$ connections. Since r_u has $O(\log n)$ ancestors, we store for u (and by that for λ) $O(\log^3 n)$ connections. Thus, the total space required is $O(n\log^3 n)$. □

We can now apply Lemma 1 to get the following theorem:

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 amortized time, and stored using $O(\varepsilon^{-1}n\log^3 n\log nN)$ space.*

5 Oracle for Undirected Graphs with Faster Update

Both Thorup [Tho04, Lemma 3.19] and Klein [Kle02] independently presented efficient vertex-vertex distance oracles for undirected planar graph that use connections sets. Klein later improved the construction time [Kle05]. They show that, in undirected planar graph, 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* [Kle05] to distinguish this type of connections from the previous one.

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 $C_G(v, Q)$ is an ε-covering set of* portals *if and only if, for every vertex t on Q there exist a vertex q on 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, and use Klein's algorithm [Kle05] to select all the portal sets $C_{G_r}(u, Q)$ 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, Ma and Ning for the static vertex-labeled case [LMN13]. 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 n pairs in $[1, n] \times \mathbb{R}$, under insertions, deletions, and Prefix Minimum Queries*

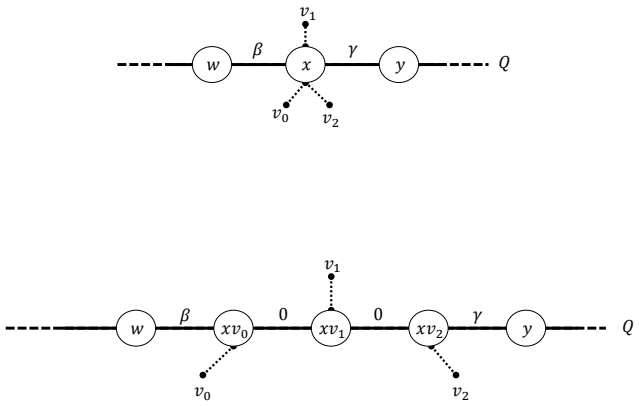


Figure 5.1: Illustration of the reduction to unique portals. Above, the path Q with the portal x that is used by v_0, v_1, and v_2. Below, x was replaced by xv_0, xv_1, and xv_2, inner connected with zero length edges. Here, xv_0, xv_1, xv_2 are the portals of v_0, v_1, and v_2 respectively. Note that this reduction does not introduce new paths in the graph, nor changes the distance along Q.

(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') 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 a set of vertices $v_0, ..., v_k$, we can split x to k copies. This does not increase $|\mathrm{G}|$ by more than a factor of ε^{-1}. See Fig. 5.1. (see figure 5.1).

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, hence the direction of Q is chosen arbitrarily. For 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 $\lambda \in \mathcal{L}_r$ 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 a dynamic sufix minimum data structure $Suf_{Q,\lambda}$ over $\{(j, h(q_j) + \delta_{G_r}(q_j, \lambda))\}_{j=0}^{k}$.

$Query(u, \lambda)$

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 13. *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 [LMN13, 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:

$$\delta_{G_r}(u, q_j) + \delta_{G_r}(q_j, t) \leq (1 + \varepsilon)(\delta_{G_r}(u, t)) \tag{5.1}$$
$$\delta_{G_r}(v, q_i) + \delta_{G_r}(q_i, t) \leq (1 + \varepsilon)(\delta_{G_r}(v, t)) \tag{5.2}$$

We add the two inequalities to get the following:

$$\delta_{G_r}(u, q_j) + \delta_{G_r}(q_j, q_i) + \delta_{G_r}(v, q_i) \leq (1 + \varepsilon)(\delta_{G_r}(u, v)) \tag{5.3}$$

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

$$Query(u, \lambda) \leq \delta_{G_r}(v, q_i) - h(q_i) + h(j) + \delta_{G_r}(u, q_j) \tag{5.4}$$
$$= \delta_{G_r}(v, q_i) + \delta_{G_r}(q_i, q_j) + \delta_{G_r}(u, q_j) \tag{5.5}$$
$$\leq (1 + \varepsilon)(\delta_{G_r}(u, v)) \tag{5.6}$$
$$\leq (1 + \varepsilon)(\delta_G(u, \lambda)) \tag{5.7}$$

Here, inequality (5.6) follows from (5.3), and (5.7) 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. □

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 14. *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 [Kle05] 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 portal 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, $x \leq \varepsilon^{-1}n$ 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 in $|\mathcal{L}|$, 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 [Wil14] 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 $\{(x, y) : l_1 \leq x \leq l_2, h_1 \leq y \leq h_2\}$. 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 theorem.

Theorem 3. *[Wil14, 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 15. *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 16 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 16. *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 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 15. □

We therefore obtain the following theorem.

Theorem 4. *For any undirected planar graph and fixed parameters ε, γ, there exists a stretch-$(1+\varepsilon)$ vertex-labeled distance oracle that approximates distances in $O(\varepsilon^{-1}\frac{\log n\log(\varepsilon^{-1}n)}{\log\log(\varepsilon^{-1}n)})$ time worst case, 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.*

6 Concluding remarks

In this work 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. The logarithmic factor can be avoided in the vertex-to-vertex case where approximate distance oracles with faster query times exist (see e.g., [Tho04, Wul16, GX15] and references therein). This is also the case for the static vertex-to-label case. For example, Mozes and Skop [MS15] presented an oracle with constant query time. Their oracle uses label connections that store the shortest distance to a labeled vertex in the entire graph. This is in contrast to our connections that only consider labeled vertices in a subgraph. Hence, their query algorithm only accesses a single node in the decomposition tree. The downside of storing distances in the entire graph is that labels changes are less contained (may affect $O(n)$ separators). It would be interesting to study whether an approximate distance oracle with logarithmic update time and $o(\log n)$ query time exists.

Another bottleneck for our directed data structure is the use of $O(\log n)$ connection sets in every node of the decomposition tree. Those sets are only used by our bottom-up update approach, which is mainly needed for removal of labeled vertices. We tried to avoid storing these sets, with no success. It remains an open question whether this costly update procedure can be avoided.

For our undirected oracle with the faster update (Section 5), we use Wilkinson's 2-sided reporting [Wil14] as a dynamic prefix/suffix minimum data structure. During our research, we tried to develop a faster dynamic prefix minimum data structure but with no success. Another interesting question that arises is whether other approaches may be used to obtain a faster prefix/suffix minimum data structure, that will lower the time bounds

of our oracle.

Bibliography

[ACD+16] Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. In *SODA*, pages 740–753. SIAM, 2016.

[ACG12] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *STOC*, pages 1199–1218. ACM, 2012.

[aOP+15] Jakub Łacki, Jakub Ocwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In *STOC*, pages 11–20, 2015.

[Che12] Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2012.

[FKS84] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.

[GX15] Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\epsilon)$-approximate distance oracle for planar graphs. In *ISAAC*, pages 625–636, 2015.

[HKRS97] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[HLWY11] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 490–501. Springer, 2011.

[KKS11] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2011.

[Kle02] Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *SODA*, pages 820–827, 2002.

[Kle05] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *SODA*, pages 146–155, 2005.

[KST13] Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *SODA*, pages 550–563. SIAM, 2013.

[LMN13] Mingfei Li, Chu Chung Christopher Ma, and Li Ning. $(1 + \epsilon)$-distance oracles for vertex-labeled planar graphs. In *TAMC*, pages 42–51, 2013.

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

[MS15] Shay Mozes and Eyal E. Skop. Efficient vertex-label distance oracles for planar graphs. In *WAOA*, pages 97–109, 2015.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *ESA*, pages 121–133, 2001.

[Som14] Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, 2014.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

[TZ01] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *STOC*, pages 183–192. ACM, 2001.

[Wil83] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space theta(n). *Inf. Process. Lett.*, 17(2):81–84, 1983.

[Wil14] Bryan T. Wilkinson. Amortized bounds for dynamic orthogonal range reporting. In *ESA*, pages 842–856, 2014.

[Wul12] Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *SODA*, pages 202–208. SIAM, 2012.

[Wul16] Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *SODA*, pages 351–362, 2016.

Appendix

A Reduction from stretch-$(1+\varepsilon)$ vertex-labeled distance oracle to scale-(α, ε) distance oracle.

We now describe how to use a scale-(ε, α) vertex-labeled distance oracle to obtain a stretch-$(1+\varepsilon)$ distance oracle. For vertex-vertex distance oracles, this reduction was proven by Thorup as captured in Lemma 1.

For the static vertex-labeled case, a similar reduction was presented by Mozes and Skop, and is as follows: The proof of Lemma 1 relies on two reductions [Tho04, Lemmas 3.2,3.8]. The first shows that from any graph G and for any $\alpha > 0$, one can construct a family of α-layered graphs $\{G_i^\alpha\}_i$ whose total size is linear in the size of G, and such that:

1. $\Sigma|G_i^\alpha| = O(|G|)$, where $|G| = |V(G)| + |E(G)|$.
2. Each $v \in V(G)$ has an index $j(v)$ s.t. any $w \in V(G)$ has $d = \delta_G(v,w) \leq \alpha$ iff $d = min\{\delta_{G^\alpha_{j(v)-2}}(v,w), \delta_{G^\alpha_{j(v)-1}}(v,w), \delta_{G^\alpha_{j(v)}}(v,w)\}$.
3. Each G_i^α is a minor of G. I.e., it can be obtained from G by contraction ad deletion of arcs and vertices. In particular, if G is planar, so is G_i^α.

Item (2.) means that any shortest path of length at most α in G is represented in at least one of three fixed graphs G_i^α. Thus, one can use scale-(α, ε) distance oracles for the α-layered graphs $\{G_i^\alpha\}$ to implement a scale-(α, ε) oracle of G.

The second reduction [Tho04, Lemmas 3.8] is a scaling argument that shows how to construct a stretch-$(1+\varepsilon)$ distance oracle for G using scale-(α, ε') distance oracles for $\alpha \in \{2^i\}_{i \in [1, \lceil \log(nN) \rceil]}$. The reduction does not rely on planarity. Now consider the vertex-labeled case. Let G^* be the

graph obtained from G by adding apices representing the labels. A vertex-to-vertex distance oracle for G^* is a vertex-labeled distance oracle for G, and vice versa. By Thorup's second reduction, it suffices to show how to construct a scale-(α,ε) vertex-vertex distance oracle for G^* for any α, ε, or, equivalently a vertex-labeled scale-(α,ε) distance oracle for G and every α, ε. Let $\alpha \in \mathbb{R}^+$. Given $u \in V(G)$ and $\lambda \in \mathcal{L}$ with $\delta_G(u,\lambda) \leq \alpha$, let $w \in V(G)$ be the closest λ labeled vertex to u. By the properties of Thorup's first reduction, there is a graph G_i^α in whitch the u-to-w distance is $\delta_G(u,w)$. Thus, a vertex-labeled distance oracle for G_i^α will report a distance of at most $\delta_G(u,\lambda)+\varepsilon\alpha$. Therefore we have the following Lemma:

Lemma 17. *For any planar graph G and fixed parameter ε, a stretch-$(1+\varepsilon)$ vertex-labeled distance oracle can be constructed using $O(\log nN)$ scale-(α,ε') vertex-labeled distance oracles where $\alpha = 2^i$, $i = 0,...\lceil \log nN \rceil$ and $\varepsilon' \in 1/2, \varepsilon/4$. Assume that the scale-$(\alpha,\varepsilon)$ vertex-labeled distance oracle supports queries in $O(T_Q(n,\varepsilon))$ and updates in $O(T_U(n,\varepsilon))$ time, and it can be constructed in $O(T_C(n,\varepsilon))$ time and uses $O(S(n,\varepsilon))$ space. There exists a $S(n,\varepsilon)\log nN$ space stretch-$(1+\varepsilon)$ vertex-labeled distance oracle that answers queries in $O(T_Q(n,\varepsilon)\log\log(nN))$ and updated in $O(T_U(n,\varepsilon)\log nN)$ time can be constructed in $O(T_C(n,\varepsilon)\log nN)$ time.*

Proof. Given a planar graph G, we decompose G to $O(\log nN)$ α-layered graphs, and for each we construct a scale-(α,ε) distance oracle. We get the space requirements, and the construction and query times by using Lemma 1. Since we must keep all $O(\log nN)$ scale oracles up to date, we perform each update operation $O(\log nN)$ times, and the lemma follows. □

תקציר

יהי G גרף אשר כל אחד מקודקודיו בעל צבע. **אוב מרחקים מקורב** לגרף בעל צביעת קודקודים, הוא מבנה נתונים, אשר בהינתן קודקוד v וצבע λ, מחזיר $(1+\varepsilon)$-קירוב למרחק בין v לקודקוד הקרוב ביותר הצבוע בצבע λ בגרף G. אוב כנ"ל ייקרא **דינאמי** אם בנוסף הוא תומך בשינוי צביעת הקודקודים.
במחקר זה נציג שלושה אובי מרחקים מקורבים דינאמים לגרף מישורי בעל צביעת קודקודים, כולם בעלי זמן שאילתה ועדכון פולילוגריתמים, ודרישות מקום קרוב ללינארי. זהו אוב המרחקים המקורב הדינאמי הראשון אשר גם זמן העדכון וגם זמן השאילתה שלו הם תת-לינאריים.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

אוב מרחקים מקורב דינאמי לגרף מישורי בעל צביעת קודקודים

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

מוגש על ידי איתי ליש
בהנחיית ד"ר שי מוזס.

אוגוסט 2017