The Interdisciplinary Center Herzliya
Efi Arazi School of Computer Science

Efficient Vertex-Label Distance Oracles For Planar Graphs

M.Sc. Dissertation

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science (M.Sc.) Research Track in Computer Science

Submitted by Eyal Skop
Under the supervision of Dr. Shay Mozes

March 2015

Acknowledgements

I would like to express several thanks to my advisor Dr. Shay Mozes. First, for introducing me to the subject of planar graphs, for without the interest in this field I would not have gone through the journey of thesis making. Second, for his guidance through rough times and hard decision makings while hitting several brick walls. Third, for investing countless hours to review, refine and condense my insights, along with personal tips. Fourth, for the passion for exploring.

Abstract

We consider distance queries in vertex labeled planar graphs. For any fixed $0 < \epsilon \leq 1/2$ we show how to preprocess a planar graph with vertex labels and edge lengths into a data structure that answers queries of the following form. Given a vertex u and a label λ return a $(1 + O(\epsilon))$-approximation of the distance between u and its closest vertex with label λ. For an undirected n-vertex planar graph the preprocessing time is $O(\epsilon^{-2} n \lg^3 n)$, the size is $O(\epsilon^{-1} n \lg n)$, and the query time is $O(\lg \lg n + \epsilon^{-1})$. For a directed planar graph with arc lengths bounded by N, the preprocessing time is $O(\epsilon^{-2} n \lg^3 n \lg(nN))$, the data structure size is $O(\epsilon^{-1} n \lg n \lg(nN))$, and the query time is $O(\lg \lg n \lg \lg (nN) + \epsilon^{-1})$.

Table of Contents

1 Introduction

Imagine you are driving your car and suddenly see you are about to run out of gas. What should you do? Obviously, you should find the closest gas station. This is the *vertex-to-label distance query problem*. Various software applications like Waze and Google maps attempt to provide such a functionality. The idea is to preprocess the locations of service providers, such as gas stations, hospitals, pubs and metro stations in advance, so that when a user, whose location is not known a priori, asks for the distance to the closest service provider, the information can be retrieved as quickly as possible.

We study this problem from a theoretical point of view. We model the network as a planar graph with labeled vertices (e.g., a vertex labeled as a gas station). We study distance oracles for such graphs. A *vertex-label distance oracle* is a data structure that represents the input graph and can be queried for the distance between any vertex and the closest vertex with a desired label. We consider approximate distance oracles, which, for any given fixed parameter $\epsilon > 0$, return a distance estimate that is at least the true distance queried, and at most $(1+\epsilon)$ times the true distance (this is known as a $(1+\epsilon)$-stretch). One would like an oracle with the following properties; queries should be answered quickly, the oracle should consume little space, and the construction of the oracle should take as little time as possible. We use the notation $\langle O(S(n))_{space}\ , O(T(n))_{time}\rangle$ to express the space requirement and query time of a distance oracle.

A dual situation is, for example, when a taxi company wants to dispatch a taxi from the station closest to the location where the taxi is required. Clearly, this problem can be solved using a *vertex-label* distance oracle by inverting the original graph.

Our results and approach We give a $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ , O(\lg\lg n + \epsilon^{-1})_{time}\rangle$ vertex-label distance oracle for *undirected* planar graphs that can be constructed in $O(\epsilon^{-2} n \lg^3 n)$ time. For directed planar graphs we give a $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}\ , O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ vertex-label distance oracle whose construction time is $O(\epsilon^{-2} n \lg^3 n \lg(nN))$. To the best of our knowledge, no non-trivial directed vertex-label distance oracles where proposed prior to the current work.

Consider a vertex-to-vertex distance oracle for a grpah with label set L. If the oracle works for general directed graphs then the vertex-to-label problem can be solved easily; add a distinct apex v_λ for each label $\lambda \in L$, and connect every λ-labeled vertex to v_λ with a zero length arc. Finding the distance from a vertex u to label λ is now equivalent to finding the distance between u and v_λ. This approach presents two main difficulties when designing efficient oracles for planar graphs. First, adding apices breaks planarity. In particular, it affects the separability of the graph. Thus, the reduction does not work with oracles that depend on planarity or on the existence of separators, which are more efficient than oracles for general graphs. Second, the reduction uses *directed* arcs, so it is unsuitable for oracles for undirected graphs. Using arcs in the reduction is crucial since connecting an apex with undirected zero length edges changes the distances in the graph. This is because the apex can be used to teleport between vertices with the same label.[1.1]

We nonetheless use this approach, and show how to overcome these obstacles. We augment a directed and an undirected variants of a distance oracle of Thorup [Tho04] for planar graphs. These oracles rely on the existence of fundamental cycle separators in planar graphs, a property that breaks when apices are added to the graph. However, we observe that once the graph is separated, Thorup's oracle does not depend on planarity. We therefore postpone the addition of the apices till a later stage in the construction of the distance oracle, when the graph has already been separated. We show that, nonetheless, approximate distances from any vertex to any label in the entire graph can be approximated. Moreover, we observe that Thorup's undirected oracle internally uses the same directed structures as in his directed oracle. It only depends on the undirectedness in making the number and sizes of these structures smaller than in the directed case. We extend this argument to handle vertex labels.

[1.1] Teleporting between vertices might be desirable in some applications. For example, calculating the walking distance between two locations without accounting traveling between train stations.

2 Related Work

We summarize related work on approximate and exact vertex-vertex distance oracles. For general graphs, no efficient (2)-stretch approximate vertex-vertex distance oracles are known to date. For any integer $k \geq 2$, Thorup and Zwick [TZ05] presented a $(2k-1)$-stretch $\langle O(kn^{1+1/k})_{space}\ , O(k)_{time}\rangle$ undirected distance oracle which is constructed in $O(kmn^{1/k})$ time. Wulff-Nilsen [Wul12] achieved the same result with preprocessing of $O(kn^{1+\frac{c}{k}})$ for universal constant c. Several more improvements of [TZ05] have been found for unweighted or sparse graphs ([BGSU08], [BK06], [BS06]).

In contrast, vertex-vertex oracles for planar graphs with stretch less then 2 have been constructed. Thorup [Tho04] gave a $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}\ , O(\lg\lg(nN)+\epsilon^{-1})_{time}\rangle$ $(1+\epsilon)$-stretch directed distance oracle, and a $\langle O(\epsilon^{-1} n \lg n)_{space}\ , O(\epsilon^{-1})_{time}\rangle$ undirected (simplified) distance oracle. Our results is based on Thorup's oracles, which are described in Section 4. Klein [Kle02] independently gave an undirected distance oracle with same bounds. Kawarabayashi, Klein and Sommer [KKS11] have shown a $\langle O(n)_{space}\ , O(\epsilon^{-2}\lg^{-2}(n))_{time}\rangle$ undirected $(1+\epsilon)$-stretch distance oracle constructed in $O(n\lg^2 n)$ time, inspired by [Tho04]. [KKS11] give a trade-off of $\langle O(\frac{\epsilon^{-1} n \lg n}{\sqrt{r}})_{space}\ , O(r+\sqrt{r}\epsilon^{-1}\lg n)_{time}\rangle|_{\forall r\leq n}$ oracle algorithms. Sommer et al. [KST13] have shown better tradeoffs for undirected oracles. For the case where $N \in poly(n)$, they achieve $\langle O^*(n\lg n)_{space}\ , O^*(\epsilon^{-1})_{time}\rangle$ oracle, where O^* hides $\lg(\epsilon^{-1})$ and $\lg^*(n)$ factors.

The vertex-to-label distance query problem was introduced by Hermelin, Levy, Weimann and Yuster [HLWY11]. For any integer $k \geq 2$, they gave a $(4k-5)$-stretch $\langle O(kn^{1+1/k})_{space}\ , O(k)_{time}\rangle$ vertex-label distance oracle (expected space) for undirected general (i.e., non-planar) graphs. This is not efficient when the number l of distinct labels is $o(n^{1/k})$. They also presented a (2^k-1)-stretch $\langle O(knl^{1/k})_{space}\ , O(k)_{time}\rangle$ undirected oracle. Chechik [Che12] improved the latter result; Second, she presents a $(4k-5)$-stretch $\langle O(knl^{1/k})_{space}\ , O(k)_{time}\rangle$ (expected space) undirected oracle.

For planar graphs, the only vertex-label distance oracle we are aware of was described by Li, Ma and Ning [LMN13]. They construct a $(1+\epsilon)$-stretch oracle with $\langle O(\epsilon^{-1} n \lg n)_{space}\ , O(\epsilon^{-1}\lg n \lg \rho)_{time}\rangle$ bounds for undirected graphs. Here, ρ is the radius of the graph, which can be $\theta(n)$. It is also shown in [LMN13] how to avoid the $\lg \rho$ factor when $\rho = O(\lg n)$. The construction time of their oracle is $O(\epsilon^{-1} n \log^2 n)$.

As for exact distance oracles, no non-trivial oracles exist for general graphs. Efficient exact oracles for planar graphs rely on small balanced separators.

Djidjev [Dji96] has given a $\langle O(S)_{space}\ , O(n^2/S)_{time}\rangle|_{\forall S\in[n,n^2]}$ and a $\langle O(S)_{space}\ , \tilde{O}(\frac{n}{\sqrt{S}})_{time}\rangle|_{\forall S\in[n^{4/3},n^{3/2}]}$ vertex-vertex distance oracle for directed planar graphs, where $\tilde{O}$ hides poly-logarithmic factor. Chen and Xu [CX00] have extended the later result range to $\langle O(S)_{space}\ , \tilde{O}(\frac{n}{\sqrt{S}})_{time}\rangle|_{\forall S\in[n^{4/3},n^2]}$. Cabello [Cab12] had given the same result but with a better preprocessing time. Wulff-Nilsen [Wul13] has given a $\langle o(n^2)_{space}\ , O(1)_{time}\rangle$ result for unweighted planar graphs. Fakcharoenphol and Rao [FR06] presented a $\langle \tilde{O}(n)_{space}\ , \tilde{O}(\sqrt{n})_{time}\rangle$ result, which has the best space-query trade-off as for today. Later results ([MW10,Kle05]) implicitly improved [FR06] result by a logarithmic factor both in preprocessing and space. Mozes and Sommer [MS12] have extended [CX00] range to $\langle O(S)_{space}\ , \tilde{O}(\frac{n}{\sqrt{S}})_{time}\rangle|_{\forall S\in[n\lg\lg n, n^2]}$. Their result is achieved by paying another $O(\sqrt{\lg n})$ factor in query time. Nussbaum [Nus11] independently gave a similar result. [MS12] also give a $\langle O(n)_{space}\ , O(n^{1/2+\epsilon})_{time}\rangle|_{\forall\epsilon>0}$ and a $\langle \tilde{O}(n)_{space}\ , \tilde{O}(\min\{l, \sqrt{n}\})_{time}\rangle$ results where l is the distance between the queried vertices, for positive edge lengths.

Tao, Papadopoulos, Sheng and Stefanidis [TPSS11] achieved a $\langle O(n)_{space}\ , O(\lg n)_{time}\rangle$ exact distance oracle over XML trees.

3 Preliminaries

A graph G is a tuple consisting of $V(G)$, a finite set of objects called *vertices* and $A(G)$, a set of *arcs*, where an arc is an ordered pair of vertices. An undirected graph is similar, but instead of $A(G)$ it consists a set of *edges* $E(G)$, where an edge is an unordered pair of vertices. Throughout this work, all graphs are *directed* unless stated otherwise. We introduce several definitions for directed graphs, which should be inferred to undirected graphs as well.

Given an arc uv, we say that u and v are the *endpoints* of uv. A vertex is *incident* to an arc if it is one of the endpoints of that arc. A u-to-v *path* is an ordered set of vertices which starts with u, ends with v, and for each consecutive vertices there is an arc from the former to the later, which are the arcs of the path. The concatenation of two paths P_1 and P_2, where last vertex of P_1 is the first vertex of P_2, is denoted $P_1 \circ P_2$.

For a path Q and a vertex set $U \subseteq V(Q)$, we define $\bar{Q}$, the *reduction* of Q to U as follows. Repeatedly applying the following procedure to Q. Let v be a vertex of Q such that $v \notin U$. Delete v from Q. If Q doesn't start with v, introduce a new arc to $A(G)$ between the other endpoints of the arcs v was incident to in Q. Also, if there are arcs lengths, add the length of the discarded arc wv to the length of the other arc of Q incident to w. Note that $|\bar{Q}| = O(|U|)$.

A *cycle* is a path that begins and ends in the same vertex. A cycle is called *simple cycle* if it contains no other cycle which is not itself. A *tree* is a set of several paths such that the union of the path's arcs in the tree, viewed as (undirected) edges, does not contain an undirected cycle, and is connected (e.g. it contains a path from each vertex to any other vertex). A tree is called *rooted* if all its paths has common starting vertex called the *root*. A *rooted spanning tree* is a rooted tree such that each vertex in $V(G)$ participates in one of the tree's paths.

Let T be a rooted spanning tree of an undirected graph G. For $u \in V(G)$, let $T[u]$ denote the unique root-to-u path in T. The *fundamental cycle* of $e \notin E(T)$ is the undirected cycle composed of $e = u_1u_2$, $E(T[u_1])$ and $E(T[u_2])$.

Let G be a graph with arc lengths. For $u, v \in V(G)$, the *distance* between u and v, denoted $\delta_G(u, v)$, is the total minimal sum of arcs lengths among all u-to-v paths. We denote by N the maximum length of an arc in G. [3.1]

Given a planar graph, a *face* is a minimal region of the plane bounded by arcs. Euler's formula asserts that for a planar graph with n vertices, m arcs, and f faces, it holds that $n - m + f = 2$. By triangulating a planar graph (e.g. adding arcs so that each face consists of at least 3 arcs) using Euler's formula shows that $|A(G)| = O(|V(G)|)$. It therefore makes sense to define $|G|$, the size of a planar graph as $|V(G)|$.

[3.1] $O(nN)$ is an upper bound on $\delta_G(\cdot, \cdot)$.

A simple cycle separates a planar graph G into an interior and exterior parts. Let G be a planar graph and T be a rooted spanning tree of G. It was shown by (Lipton and Tarjan [LT79]) how a fundamental cycle C of T can be found so that the strict interior and exterior of the graph with respect to C are roughly the same size. This graph property is strongly used in our result.

Let $L = \{\lambda_i\}_{i=1}^{l}$ be a set of l labels. A vertex-labeled graph is a graph G equipped with a function $f : V(G) \to L$. We define $V_\lambda = \{v \in V(G) | f(v) = \lambda\}$ to be set of vertices with label λ. For a vertex-labeled G and $\lambda \in L$, we define $\delta_G(u, \lambda) = \min\limits_{w \in V_\lambda} \delta_G(u, w)$ to be the distance in G from u to the closest λ-labeled vertex.

A *vertex-label distance oracle* is a data structure that, given a vertex $v \in V$ and a label $\lambda \in L$, outputs an (approximation) of $\delta_G(v, \lambda)$. We note that this problem is a generalization of the basic distance oracle problem (in which each vertex is given a unique label). Constructing an $O(nl)$-space vertex-label distance oracle is trivial, by precomputing and storing the distance between each vertex and each possible label. The goal is, therefore, to devise an oracle which requires substantially less than nl space, while allowing for fast queries.

4 Thorup's Approximate Distance Oracle

In this section we outline the distance oracle of Thorup [Tho04]. This description is not new, but is necessary for understanding our results. Our description does not go into all the details of Thorup's oracle, but rather focuses on those that are not used by as black box.

Thorup shows that the problem of constructing a distance oracle for a directed graph can be reduced to constructing a distance oracle for a restricted kind of graph, defined in the following.

Definition 4.1. *A set T of arcs in a graph H is a (t, α)-layered spanning tree if it satisfies the following properties:*

- *Disoriented - it can be oriented to form a spanning tree of H.*
- *Each branch (a path from the root of T) is a concatenation of at most t shortest paths in H.*
- *Each shortest path in a branch of T is of length at most α*

Definition 4.2. *A graph H is called (t, α)-layered if it has a (t, α)-layered spanning tree.*

Definition 4.3. *A scale-(α, ϵ) distance oracle for a (t, α)-layered graph H is a data structure that, when queried for $\delta_H(v, w)$ returns*

$$d(v,w) = \begin{cases} d \in [\delta_H(v,w), \delta_H(v,w) + \epsilon\alpha] & \delta_H(v,w) \leq \alpha \\ \infty & otherwise \end{cases}$$

The reduction is summarized in the following lemma. See Appendix A for details.

Lemma 4.1. *([Tho04, Sections 3.1,3.2,3.3]) Given a scale-(α, ϵ') $\langle O(s(n,\epsilon'))_{space}$, $O(t(\epsilon'))_{time}\rangle$ algorithm, a $(1+\epsilon)$-stretch $\langle O(s(n,\epsilon)\lg(nN))_{space}$, $O(t(\frac{1}{4})\lg\lg(nN) + t(\frac{\epsilon}{4}))_{time}\rangle$ algorithm can be constructed to any input graph.*

Thorup shows each graph can be decomposed to a $(3, \alpha)$-layered graphs, of total linear size (with α the distance bound of the graph). Therefore its suffices to show how to construct a scale-(α, ϵ) distance oracle.

The remainder of the section describes Thorup's scale-(α, ϵ) oracle.

4.1 The frame structure

The construction is recursive, using shortest path separators.

Lemma 4.2. *(Fundamental Cycle Separator [LT79]) Let H be an undirected planar graph with a rooted spanning tree T and function w assigning non-negative weights to edges. One can find an edge $e \notin T$ such that neither the weight strictly enclosed by the fundamental cycle of e nor the weight not enclosed by the fundamental cycle of e exceeds $\frac{2}{3}$ the weight of H.*

A planar graph G can be decomposed by computing a shortest path tree for an arbitrary vertex, and applying lemma 4.2 recursively. Choosing the spanning tree in lemma 4.2 to be a shortest path tree guarantees that each fundamental cycle separator found consists of two shortest paths. The decomposition can be represented by a binary tree $\mathcal{T}$ in the following manner. [4.1] See fig. 4.1 in for an illustration.

- Each node r of $\mathcal{T}$ is associated with a subgraph G_r of G. The subgraph associated with the root of $\mathcal{T}$ is all of G.
- Each non-leaf node r of $\mathcal{T}$ is associated with the fundamental cycle separator S_r found by invoking lemma 4.2 on G_r.
- Each non-leaf node r has two children, whose associated subgraphs are the interior and exterior of S_r. The vertices and edges of the separator belong to both subgraphs.

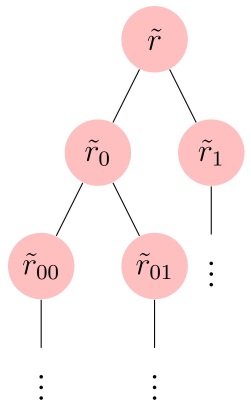

Fig. 4.1: An illustration of the decomposition tree $\mathcal{T}$. The root $\tilde{r}$ is associated with $G_{\tilde{r}} = G$. The children of $\tilde{r}$, $\tilde{r}_0$ and $\tilde{r}_1$, are associated with the interior and exterior of the separator $S_{\tilde{r}}$ of $G_{\tilde{r}}$.

Let r be a node of $\mathcal{T}$. The *frame* F_r of G_r is the set of (shortest) paths in $\bigcup_{r'}(S_{r'} \cap G_r)$, where the union is over strict ancestors r' of r in $\mathcal{T}$. Each non-leaf node r stores its frame F_r. A standard argument shows that, by alternating the separation criteria between number of edges in the graph and number of paths in the frame, one can get frames consisting of a constant number of paths (Appendix B).

[4.1] We refer to the vertices of $\mathcal{T}$ as *nodes* to distinguish them from the vertices of the graph G.

For $r \in \mathcal{T}$, let G_r° denote the subgraph of $G_r \setminus F_r$. That is, G_r° is the graph obtained from G_r by removing the arcs of the frame F_r as well as any vertices of F_r that become isolated as a result of the removal. The sizes of the G_r°'s decrease by a constant factor along $\mathcal{T}$, while the sizes of the G_r's need not because there is no bound on the size of the fundamental cycle in lemma 4.2. This may pose a problem, since the frame F_r is stored by every node r. To overcome this, the algorithm stores the reduction of F_r to G_r° instead of F_r itself.

4.2 Thorup's scale-(α, ϵ) algorithm [Tho04]

The main idea is to store just a subset of the pairwise distances in the graph, from which all distances can be approximately computed efficiently. Given a $(3, \alpha)$-layered graph H and a shortest path $Q \in H$, Thorup shows that for every vertex $v \in H$, there exists a set $C(v, Q)$ $(C(Q, v))$ of $O(\epsilon^{-1})$ vertices on Q, called *connections*, such that the distances (called *connection lengths*) from (to) every vertex of H to (from) its connections on Q can be used to approximate, in $O(\epsilon^{-1})$ time, the length of any shortest path from/to v in H that intersects Q. Thorup essentially proves the following lemma whose proof is in Appendix C, lemma C.1:

Lemma 4.3. *Let Q be a shortest path in a $(3, \alpha)$-layered graph G. Assume a shortest u-to-w path P in G intersects a shortest path Q. There exists connections sets $C(u, Q)$ and $C(Q, w)$ s.t.*

$$\delta_{G_Q^{uw}}(u, w) \leq \delta_G(u, w) + 2\epsilon\alpha \tag{4.1}$$

where G_Q^{uw} is a graph with vertices u, w, the vertices of the reduction of Q to the connections of u and w, and with u-to-Q and Q-to-v arcs whose lengths are the corresponding connection lengths of $C(u, Q)$ and $C(Q, w)$.

For efficiency reasons, instead of storing exact connection lengths $\delta(\cdot, \cdot)$, the algorithm will compute approximate connection lengths.

A *Lowest Common Ancestor* (LCA) data structure for a tree T is a data structure that, given any two nodes x, y of T, returns the node furthest from the root that is a parent of both x and y. There exist LCA data structures of linear size and constant query time by Harel and Tarjan [HT84].

Let u, w be vertices of G. Let r_u, r_w be the leaves of $\mathcal{T}$ such that $u \in G_{r_u}$ and $w \in G_{r_w}$. Let r be the LCA of r_u and r_v in $\mathcal{T}$. Observe that u and w are separated by S_r. Hence, every u-to-w path in G must intersect S_r. However, a u-to-w path may or may not intersect F_r. See fig. 4.2.

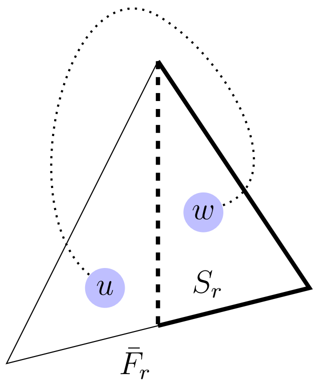

Fig. 4.2: The solid lines (thin and thick) indicate $\bar{F}_r$, the reduced frame of G_r. The bold lines (solid and dashed) indicate S_r, the separator of G_r. Vertices u and w are vertices of G_r° separated by S_r. Every u-to-w path must intersect S_r. The dashed line shows a possible shortest u-to-w path in G.

Suppose first that a shortest u-to-w path P (in G) does intersect F_r. We write $P = P_0 \circ P_1$. Path P_0 is a maximal prefix of P whose internal vertices belong to G_r°. We call this kind of paths *type-0* paths. Note that type-0 paths start at a vertex of G_r°, end at a vertex of F_r and are confined to G_r°. Path P_1 consists of the remainder of P, and is referred to as a *type-1* path. Note that type-1 paths start at a vertex of $F_r \cap G_r^\circ$, end at a vertex of G_r°, but are *not* confined to G_r°. It is not difficult to convince oneself that, to be able to approximate $\delta_G(u, w)$, it suffices to keep, for every $Q \in F_r$, connections $C(u, Q)$ of type-0 (i.e. the connection lengths are relative to G_r°, not the entire G) and connections $C(Q, w)$ of type-1 (i.e. the connection lengths are relative to the entire graph G). For details, see Appendix C, lemma C.3.

Now suppose that no shortest u-to-w path P (in G) intersects F_r. Then every u-to-w path P (in G) intersects S_r and is confined to G_r°. Then, to approximate P it suffices to keep, for every $Q \in S_r$, type-0 connections $C(u, Q)$ and $C(Q, w)$.

The distance oracle therefore keeps, for each $r \in \mathcal{T}$, for each vertex $u \in G_r^\circ$:

1. connections $C(u, Q)$ of type-0 for all $Q \in F_r$.
2. connections $C(Q, u)$ of type-1 for all $Q \in F_r$.
3. connections $C(u, Q)$ and $C(Q, u)$ of type-0 for all $Q \in S_r$.

These connections, over all $u \in G_r$ and all paths in $S_r \cup F_r$ are called the (type-0 or type-1) connections of r. In addition, the data structure stores:

– A mapping of each vertex $v \in G$ to a leaf node $r_v \in \mathcal{T}$ s.t. $v \in G_{r_v}$.
– A least common ancestor data structure over $\mathcal{T}$.

The space bottleneck is the size of the sets maintained. Each vertex v belongs to G_r° for $O(\lg n)$ nodes r of $\mathcal{T}$. For each of the $O(1)$ paths in the frame and separator of each such node r, v has a set of $O(\epsilon^{-1})$ connections. Hence the total space required by Thorup's oracle is $O(\epsilon^{-1} n \lg n)$.

We next describe how a query is performed. Given a u-to-w distance query, let r be the least common ancestor of r_u and r_w in $\mathcal{T}$. The algorithm computes, for each path Q of $S_r \cup F_r$ the length of a shortest u-to-w path that intersects Q using the connections $C(u,Q)$ and $C(Q,w)$ (of both type 0 and type 1). By construction of $\mathcal{T}$, the number of such paths Q is constant. It is easy to see that computing the distance estimate for each Q can be done in $O(\epsilon^{-1})$ time. Thus, an $(1+\epsilon)$-approximate distance is produced in $O(\epsilon^{-1})$ time.

The Construction Algorithm We now mention some, but not all the details of Thorup's $O(\epsilon^{-2} n \lg^3 n)$-time construction algorithm. Refer to the appendices and to [Tho04, subsection 3.6] for the full details.

The computation of the connections and connection lengths is done top-down the decomposition tree $\mathcal{T}$. Appendix C describes a divide-and-conquer procedure of Thorup that constructs the connections $C(u,Q)$ for all vertices u in a graph H, and a single shortest path Q. A symmetric procedure computes $C(Q,u)$. We summarize the procedure in the following lemma.

Let H be a graph. Let Q be a shortest path in H. Let $sssp(Q,H)$ be the smallest number s.t. for any subgraph H_0 of H, and any vertex $q \in Q_0$, where Q_0 is the reduction of Q to H_0, we can compute single source shortest paths from q in the graph $Q_0 \cup H_0$ in $O(sssp(Q,H)|E(H_0)|)$ time. It is easy to see that a standard implementation of Dijkstra's algorithm with priority queues implies $sssp(Q,H) = O(\lg |E(H)|)$. If H is planar, then $sssp(Q,H) = O(1)$ by Henzinger et al. [HKRS97].

Lemma 4.4. *Let Q be a shortest path in a directed graph H. For every $u \in H$, there exists a set of $O(\epsilon^{-1})$ connections such that by only recording the connection lengths $C(u,Q), C(Q,u)$ between every vertex in H and its connections, one can approximate, for every two vertices $w, u \in H$, the length of a shortest u-to-w path that intersects Q with additive $2\epsilon\alpha$ error in $O(\epsilon^{-1})$ time. All connections and connection lengths can be computed in $O(\epsilon^{-1} sssp(Q,H)|E(H)| \lg |V(Q)|)$ total time.*

Naively using lemma 4.4 on G_r° for all $r \in \mathcal{T}$ is efficient, but only generates type-0 connections on S_r. Using lemma 4.4 on G_r would produce type-0 connections on F_r, but is not efficient since $|F_r|$ can be much larger than $|G_r^\circ|$. Instead, For each path Q in $S_r \cup F_r$, the algorithm uses the reduction $\bar{Q}$ of Q to the vertices of Q that belong to G_r°. Let G_r^Q be the graph composed of G_r° and $\bar{Q}$. Note that $|G_r^Q| = O(|G_r^\circ|)$. The type-0 connections on $S_r \cup F_r$ can now be computed by applying lemma 4.4 to G_r^Q.

It remains to compute type-1 connections. Recall that these connection lengths reflect distances in the entire graph, not just in G_r. Clearly, applying lemma 4.4 on G for every r

is inefficient. Instead, the computation is done top-down $\mathcal{T}$, augmenting G_r° with the connection lengths of ancestors of r in $\mathcal{T}$, which have already been computed, and represent distances outside G_r. This is done as follows.

Lemma 4.5. *Let $r \in \mathcal{T}$. Type-1 connections for r can be computed using just G_r and all type-0 connections of strict ancestors of r.*

Proof. Let Q be a path in F_r. Let X_r^Q be the graph composed of:

- The vertices of G_r°
- The vertices and arcs of $\bar{Q}$, the reduction of Q to the vertices of Q which belong to G_r°.
- For each strict ancestor r' of r, for each path $Q' \in S_{r'}$, the vertices and arcs of $\bar{Q}'$, the reduction of Q' to vertices of Q' with (type-0) connections to vertices of G_r° or with (type-0) connections from vertices of $\bar{Q}$, along with arcs representing the corresponding connection lengths of r'.

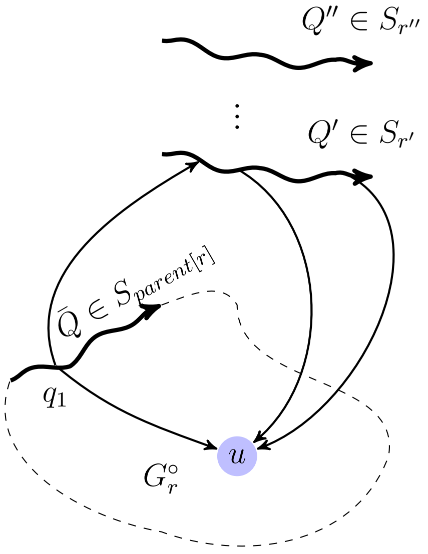

Fig. 4.3: The figure illustrates a part of X_r^Q. In this example Q is a path in the separator of the parent of r (in general Q is a path in F_r, so it may belong to the separator of an ancestor of r). The vertices of G_r° are enclosed in F_r, which is represented by the dashed lines and by $\bar{Q}$. Paths from $\bar{Q}$ to λ-labeled vertices such as u_1 and u_2 are represented in X_r^Q by arcs between $\bar{Q}$ and λ. These arcs correspond to the type-0 connections of λ on $\bar{Q}$ in the parent of r. All solid arcs are part of X_r^Q. A shortest path from $q_1 \in \bar{Q}$ to $u \in G_r^\circ$ might be enclosed in $G_{parent[r]}^\circ$ (type-0 $C(\bar{Q}, u)$) or approximated through a path which crosses a separator of an ancestor of r (in the figure, it is composed of arcs from $C(q_1, Q')$, Q' and $C(Q', u)$).

Since each vertex of G_r° has $O(\epsilon^{-1})$ connections to each path in the frame of each of r's $O(\lg n)$ ancestors, the size of X_r^Q is $O(\epsilon^{-1} \lg n |V(G_r^\circ)|)$.

We now describe why X_r^Q yields the desired type-1 connections. We explain that for $r \in \mathcal{T}$, X_r^Q $2\epsilon\alpha$-approximates the distance in G from vertices of F_r to vertices of G_r°. Hence applying lemma 4.4 on X_r^Q approximates distances in G with additive error $3\epsilon\alpha$.

The crucial point is to show that the additive error does not accumulate along the recursion. We prove that for any $Q \in F_r$, and any $q \in Q$ and $u \in G_r^\circ$, $\delta_{X_r^Q}(q,u) \leq \delta_G(q,u) + 2\epsilon\alpha$.

Let r be a node of $\mathcal{T}$. If r is the root of $\mathcal{T}$, F_r is empty, so the lemma holds vacuously since no type-1 connections are computed. Otherwise, let Q be a path in F_r, $q \in Q$, and $u \in G_r^\circ$. Consider a shortest q-to-u path P. Let r' be the rootmost ancestor of r such that P intersects $S_{r'}$. Let $\tilde{Q}$ be a path of $S_{r'}$ intersected by P. Consider the type-0 connection lengths from q to $\tilde{Q}$ in r' and the connections from $\tilde{Q}$ to u. These connection lengths were calculated with respect to exact distances in $G_{r'}^\circ$. Since $\tilde{Q}$, $C(q,\tilde{Q})$ and $C(\tilde{Q},u)$ are all present in X_r^Q, it follows from lemma 4.3 that $\delta_{X_r^Q}(q,u) \leq \delta_{G_{r'}^\circ}(q,u) + 2\epsilon\alpha$. Since P is confined to $G_{r'}^\circ$, $\delta_{G_{r'}^\circ}(q,u) = \delta_G(q,u)$, and the lemma follows.

Lemma 4.6. *For a node $r \in \mathcal{T}$, the computation of lemma 4.5 requires $O(\epsilon^{-2}|V(G_r^\circ)|\lg^2 n)$ total time.*

Proof. One can implement a simple shortest paths algorithm in X_r^Q with $sssp(\bar{Q}, X_r^Q) = O(1)$; first, relax all outgoing arcs of $\bar{Q}$. Second, for each reduced path $\bar{Q}'$ of a frame of an ancestor of r, relax the arcs of $\bar{Q}'$ by their order along $\bar{Q}'$. Third, relax $\bar{Q}'$-to-G_r° arcs for any former $\bar{Q}'$.

X_r^Q has $O(\epsilon^{-1}|V(G_r^\circ)|\lg n)$ arcs and vertices because all ancestral separator paths are in reduced form. Hence, applying lemma 4.4 to X_r^Q for any $Q \in F_r$ requires $O(\epsilon^{-2}|V(G_r^\circ)|\lg^2 n)$ time. Since there are constant number of paths in F_r, it is the total runtime.

Lemma 4.7. *Given a $(3,\alpha)$-layered graph, the construction algorithm runtime is $O(\epsilon^{-2} n \lg^3 n)$.*

Proof. Type-0 connections are computed by applying lemma 4.4, for any $Q \in S_r \cup F_r$, to G_r^Q using standard dijkstra algorithm. [4.2] Since $|E(G_r^Q)| = O(|E(G_r^\circ)|)$, this takes $O(\epsilon^{-1}|E(G_r^\circ)|\lg^2 n)$ time. Type-1 connections are computed by applying lemma 4.5 to X_r^Q, which takes $O(\epsilon^{-2}|V(G_r^\circ)|\lg^2 n)$ by lemma 4.6.

Hence, computing all connections for a single $r \in \mathcal{T}$ takes $O(\epsilon^{-2}|V(G_r^\circ)|\lg^2 n)$. Summing over all $r \in \mathcal{T}$, the total preprocessing time is $O(\epsilon^{-2} n \lg^3 n)$.

By lemma 4.1 and lemma 4.7, we obtain the following theorem:

Theorem 4.1. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}, O(\lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ distance oracle can be constructed in $O(\epsilon^{-2} n \lg^3 n \lg(nN))$ time.*

[4.2] Since G_r is planar, one may use [HKRS97] instead to achieve $O(\epsilon^{-1}|E(G_r^Q)|\lg n)$ complexity. However this is not the bottleneck.

For the undirected case, [Tho04] states a version of lemma 4.4 that can be used without the reduction to $(3, \alpha)$-layered graphs. The statement of the lemma differs from lemma 4.4 in the error term of the approximation.

Lemma 4.8. *Let Q be a shortest path in an undirected graph H. For every $u \in H$, there exists a set of $O(\epsilon^{-1})$ connections such that by only recording the connection lengths $C(u, Q)$ between every vertex in H and its connections, one can approximate, for every two vertices $w, u \in H$, the length $\delta_H(u, w)$ of a shortest u-to-w path that intersects Q with additive $\epsilon\delta_H(u, w)$ error in $O(\epsilon^{-1})$ time. All connections and connection lengths can be computed in $O(\epsilon^{-1} sssp(Q, H)|E(H)| \lg |V(Q)|)$ total time.*

In Appendix D we discuss how the directed case lemmas (described in Appendix C) are altered as to establish lemma 4.8 for the undirected case. The discussion of the undirected case in Thorup's paper [Tho04] is sketchy at places since many parts of the undirected case are essentially identical to the directed case. Indeed, one deduction step seems to fail for the undirected case. We show in Appendix D how to correct this flaw, so that lemma 4.8 is correct. Lemma 4.8 implies better bounds for undirected graphs.

Theorem 4.2. *A $(1 + \epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\epsilon^{-1})_{time}\rangle$ distance oracle for an* undirected *planar graph can be constructed in $O(\epsilon^{-2} n \lg^3 n)$ time.*

5 Efficient Directed Approximate Vertex-Label Distance Oracle

The idea is to adapt Thorup's oracle (section 4) to the vertex-label case. The reduction introduced in section 4 to reduce the problem to $(3, \alpha)$-layered graphs applies to the vertex-labeled case as well. Given a labeled graph, for each label λ we add an apex, along with zero length arcs from all λ-labeled vertices to λ's apex. Since the reduction holds for any graph, it holds for the described auxiliary graph, and hence to the vertex-labeled case. Therefore, it suffices to construct a scale-(α, ϵ) oracle as in the vertex-vertex case.

Thorup's oracle supports one-to-one (vertex-to-vertex) distance queries, whereas here we need one-to-many distance queries. Given two vertices u, v, Thorup's oracle finds the lowest common ancestor (LCA) of r_u and r_v in $\mathcal{T}$, and uses its connections to produce the answer. In a one-to-many query, there is no analogue for v. We do know, however, that a shortest u-to-λ path must intersect the separator of the leafmost node r in $\mathcal{T}$ that contains u and some λ-labeled vertex. The node r takes the role of the LCA of r_u and r_v. In order to be able to use r's connections in a distance query one must make sure that r's connections represent approximate distances to λ-labeled vertices in the entire graph, not just in G_r°.

We define a set $\mathcal{L}$ of new (artificial) vertices, one per label. For every $r \in \mathcal{T}$, let $\mathcal{L}_r$ be the restriction of $\mathcal{L}$ to labels in G_r°. Namely, $\mathcal{L}_r = \{\lambda \in \mathcal{L} | G_r^\circ \cap V_\lambda \neq \emptyset\}$. Let $\hat{G}_r^\circ$ be the graph with vertex set $V(\hat{G}_r^\circ) = V(G_r^\circ) \cup \mathcal{L}_r$ whose arcs are the arcs of G_r° along with a zero-length arc from each λ-labeled vertex of G_r° to the corresponding vertex in $\mathcal{L}_r$. Note that the number of vertices and arcs is increased by no more than a constant factor.

For every $r \in \mathcal{T}$ and $\lambda \in \mathcal{L}_r$, the oracle stores connections w.r.t. λ of both type-0 and type-1. For a path $Q \in S_r \cup F_r$, the type-0 connections $C(Q, \lambda)$ are connections in the graph obtained from G_r° by adding an artificial vertex λ, along with zero length arcs from all λ-labeled vertices in G_r° to λ. The type-1 connections $C(Q, \lambda)$ are connections in the graph obtained from G by adding an artificial vertex λ, along with zero length arcs from all λ-labeled vertices in G to λ. Before explaining how to compute these connections we discuss how a distance query is performed.

Obtaining the distance from u to λ is done by finding the lowest ancestor r of r_u with $\lambda \in \mathcal{L}_r$. A shortest u-to-λ path must cross S_r, and perhaps also F_r. The algorithm estimates, for each path Q of $S_r \cup F_r$, the length of a shortest u-to-λ path that intersects Q, using the connections $C(u, Q)$ and $C(Q, \lambda)$ stored for r (Since $\lambda \in \mathcal{L}_r$, r does store Q-to-λ connections).

Finding r can be done by binary search on the path from r_u to the root of $\mathcal{T}$. The number of steps of the binary search is $O(\lg \lg n)$. Finding whether a node r' has a vertex with label λ can be done, e.g., by storing all unique labels in $G_{r'}^\circ$ in a binary search tree, or by hashing. In the former case finding r takes $O(\lg \lg n \lg |L|)$ time, and in the latter $O(\lg \lg n)$, assuming the more restrictive word-RAM model of computation.

The Construction Algorithm It remains to show how the connections are computed. We begin with the type-0 connections. For every $r \in \mathcal{T}$, for every $Q \in S_r \cup F_r$, the algorithm computes connections on Q w.r.t. each vertex of $\hat{G}^\circ_r$ by invoking lemma 4.4 to $\hat{G}^\circ_r$.

As for the quality of approximation, we must show that the connection lengths to the artificial vertices are useful for approximate distance queries. For the type-0 connections, the $\epsilon\alpha$-approximation is immediate because the desired distances are in $\hat{G}^\circ_r$.

We now show how to compute the type-1 connections without invoking lemma 4.8 to the entire input graph G at every call. The crucial point is that the connections to the artificial vertex λ from separators of ancestors of r represent distances to vertices with label λ that are not necessarily in G°_r.

Lemma 5.1. *Let $r \in \mathcal{T}$. Type-1 connections of r to label $\lambda \in \mathcal{L}_r$ can be computed using just the (type-0) connections of strict ancestors of r. Computing all type-1 connections to $\mathcal{L}_r$ for all $r \in \mathcal{T}$ can be done in $O(\epsilon^{-2} n \lg^3 n)$.*

Proof. Let Q be a path in F_r. Let $\overrightarrow{X^Q_r}$ be the graph composed of the following: (see fig. 5.1 for an illustration)

- The vertices $\mathcal{L}_r$
- The vertices and arcs of $\bar{Q}$, the reduction of Q to the vertices of Q which belong to G°_r.
- For each strict ancestor r' of r, for each path $Q' \in S_{r'}$, the vertices and arcs of $\bar{Q}'$, the reduction of Q' to vertices that are (type-0) connections (in $G^\circ_{r'}$) of Q' w.r.t. vertices in $Q \cup \mathcal{L}_r$, along with arcs representing the corresponding connection lengths.

Let r be a node of $\mathcal{T}$, Q be a path in F_r, $q \in \bar{Q}$ and $\lambda \in \mathcal{L}_r$. We show that $\overrightarrow{X^Q_r}$ approximates the distance from every q to its closest λ-labeled vertex with an additive error of $2\epsilon\alpha$, namely:

$$\delta_{\overrightarrow{X^Q_r}}(q, \lambda) \leq \delta_G(q, \lambda) + 2\epsilon\alpha \tag{5.1}$$

If r is the root of $\mathcal{T}$, F_r is empty, so the lemma holds vacuously since no type-1 connections are computed. Otherwise, let u_λ be a closest λ-labeled vertex to q. Consider a shortest q-to-λ path P with an endpoint u_λ. Let r' be the rootmost ancestor of r such that P intersects $S_{r'}$. Let $\tilde{Q}$ be the first path of $S_{r'}$ that intersects P. Consider the type-0 connection lengths from q to $\tilde{Q}$ and from $\tilde{Q}$ to λ in r'. These lengths were calculated with respect to exact distances in $\hat{G}^\circ_{r'}$. Since $\tilde{Q}$, $C(q, \tilde{Q})$ and $C(\tilde{Q}, \lambda)$ are all present in $\overrightarrow{X^Q_r}$, it follows that $\delta_{\overrightarrow{X^Q_r}}(q, \lambda) \underset{\text{lemma 4.3}}{\leq} \delta_{\hat{G}^\circ_{r'}}(q, \lambda) + 2\epsilon\alpha$. Since P is confined to $G^\circ_{r'}$, $\delta_{\hat{G}^\circ_{r'}}(q, \lambda) = \delta_{G^\circ_{r'}}(q, u_\lambda) = \delta_G(q, \lambda)$, and the lemma follows.

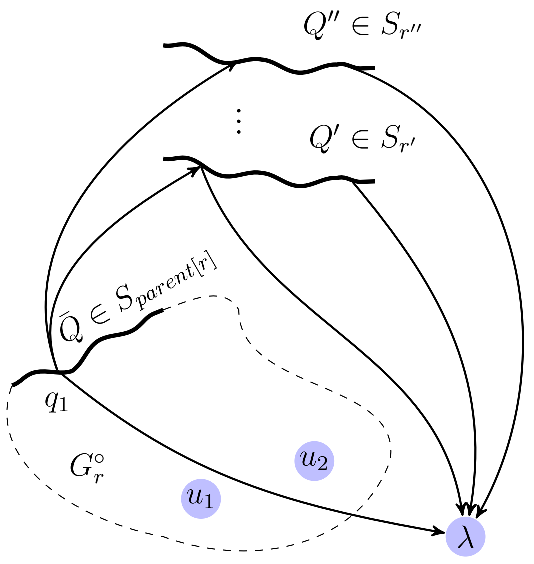

Fig. 5.1: The figure illustrates a part of X_r^Q for the labels case, similarly to fig. 4.3. The vertices u_1 and u_2 are λ-labeled vertices of G_r°, and are not part of X_r^Q. Paths from $\bar{Q}$ to λ-labeled vertices such as u_1 and u_2 are represented in X_r^Q by arcs between $\bar{Q}$ and λ. These arcs correspond to the type-0 connections of λ on $\bar{Q}$ in the parent of r. All solid arcs are part of X_r^Q. A shortest path from $q_1 \in \bar{Q}$ to $\lambda \in \mathcal{L}_r$ is approximated by connections from q_1 to a separator of an ancestor of r and from there to λ. Note that $C(Q', \lambda)$ represent distances from Q' to a λ-labeled vertex which is not necessarily in G_r°.

The construction algorithm differs from the one in subsection 4.2 in the existence of the artificial vertices. Since $|\mathcal{L}_r| = O(G_r^\circ)$ for any $r \in \mathcal{T}$, the total running time and space requirements remains as in subsection 4.2.

By the reduction and the modified construction and query, the following theorem follows:

Theorem 5.1. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}, O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ Vertex-Label Distance Oracle can be constructed in $O(\epsilon^{-2} n \lg^3 n \lg(nN))$ time w.h.p.* [5.1] *for a directed planar graph with n vertices and maximum arc length N.*

[5.1] The probability in the construction time is only due to the use of perfect hashing.

6 Efficient Undirected Approximate Vertex-Label Distance Oracle

As mentioned at the end of section 4, one can get a more efficient distance oracle for undirected planar graphs by using lemma 4.8 and avoiding the reduction to $(3, \alpha)$-layered graphs. We show that the same applies to vertex-labeled distance oracles.

In the directed case, the algorithm in section 5 adds artificial vertices to G_r° (for each r) in order to represent connections to labels.

However, in the undirected case, doing so is bound to fail. To see why, suppose vertices u and v both have label λ. Adding an artificial vertex λ and zero-length undirected edges $v\lambda$ and $u\lambda$ creates a zero-length path between u and v that does not exist in the original graph. While this does not change the distance between any vertex and its closest λ-labeled vertex, it may change distances between a vertex and its closest λ'-labeled vertex ($\lambda' \neq \lambda$). Therefore, we would have liked to add, for each label λ *separately*, a single artificial vertex λ, and compute the connection sets $C(Q, \lambda)$. Doing so would result in correct distance estimates, but is not efficient. We show how to compute the connections $C(Q, \lambda)$ without actually performing this inefficient procedure. Instead of having a single artificial vertex per label, it is split into many artificial vertices (one for each incident edge). The problem with this approach is that the number of connections becomes too large (each split vertex has its own set of $O(\epsilon^{-1})$ connections). We employ a procedure of Thorup, originally used by him to achieve efficient construction, to select a small subset of these connections and still get the desired approximation.

6.1 Connections sets for undirected graphs

In order to explain the algorithm for the undirected case, we need to elaborate on the properties of the sets of connections used to obtain the approximation. We did not go into these details in the description of the directed case since there we could use the lemmas of Thorup as black boxes.

We use the notation $\ell(\cdot, \cdot)$ to represent connection lengths. The connections sets generated by lemma 4.4 and lemma 4.8 are ϵ-*covering* sets. $C(Q, v)$ is an ϵ-covering set of Q with respect to vertex v (in the undirected case) if for each $q \in Q$ there is some $q^* \in C(Q, v)$ such that $\delta(q, q^*) + \ell(q^*, v) \leq (1 + \epsilon)\delta(q, v)$. We say that the vertex q^* ϵ-*covers* q (with respect to v).

We present lemmas which are adaptations of Thorup's lemmas for the directed case. These lemmas are implicit in [Tho04]. We also present their extensions to the labeled case. The first lemma establishes the query approximation bound for the undirected ϵ-cover definition (stated less precisely in lemma 4.3 for the directed case). The second is a variant of Thorup's *thinning procedure*, and lemma 6.3 is its extension to the vertex-labeled case.

Lemma 6.1. *([Kle02, lemma 4.1]) Let u, w be vertices in an undirected graph G. Let Q be a shortest path in G such that a u-to-w shortest path intersects Q. Let G_Q^{uw} be the graph composed of u, w, Q and edges induced by ϵ-covering sets $C(u, Q)$ and $C(Q, w)$.*

$$\delta_{G_Q^{uw}}(u, w) \leq (1 + \epsilon)\delta_G(u, w) \tag{6.1}$$

The proof of lemma 6.1 is similar to the directed case and illustrated in fig. 6.1. See Appendix C, lemma C.1.

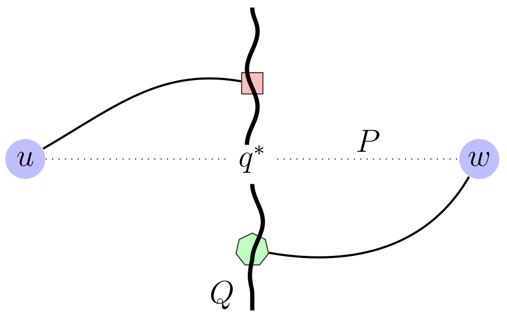

Fig. 6.1: P (dotted in the figure) is a u-to-w shortest path which intersects Q at q^*. The edges from u, w to Q connect them to their connections on Q which ϵ-cover q^*. All solid edges are part of G_Q^{uw}. The u-to-w solid edges path in the image approximates the distance of P in G_Q^{uw}.

Let G_λ be the graph obtained from G by adding an artificial vertex λ, along with zero length edges from all λ-labeled vertices in G to λ. Let $C(Q, \lambda)$ be an ϵ-cover of Q with respect to λ in G_λ. Note that lemma 6.1 claim is also true with respect to G_λ with λ as one of the vertices.

An ϵ-cover is called *clean* if it is inclusion-wise minimal, see Appendix C for details. Let $Q = (q_0, \ldots, q_{k-1})$ be a path. Keeping $\delta_Q(q_0, q_i)$ for $i = 0 \ldots k-1$ enables computing $\delta_Q(q_i, q_j)$ for $i \leq j$ in constant time. For a vertex u, a path Q and connection $q \in C(Q, u)$, we denote $\ell(q, u)$ the connection length of q w.r.t. vertex u. Next, we describe the thinning lemma which, given a large ϵ-cover, computes a small ϵ'-cover. Since the set of distances between a vertex u and all vertices of Q is a trivial ϵ-cover (even for $\epsilon = 0$), the lemma implies a small ϵ-cover always exists.

Lemma 6.2. *Let Q be a path in an undirected graph G, and let v be a vertex. Let $D(Q, v)$ be an ordered ϵ_0-cover. For any $\epsilon_1 \leq 1$, a clean and ordered $(2\epsilon_0 + \epsilon_1)$-cover $C(Q, v) \subseteq D(Q, v)$ of size $O(\epsilon_1^{-1})$ can be constructed in $O(|D(Q, v)|)$ time.*

Proof. The proof is constructive. Let $(\bar{q}, v)$ be a connection with minimal connection length in $D(Q, v)$. The vertex $\bar{q}$ splits Q into two subpaths, Q_0 and Q_1 . For each $Q' \in \{Q_0, Q_1\}$, the algorithm operates as follows. First, it adds $(\bar{q}, v)$ to $C(Q', v)$. The algorithm will now progress towards the other endpoint of Q'. We say $\tilde{q}$ *semi ϵ-covers* q^*

if $\delta_Q(q^*, \tilde{q}) + \ell(\tilde{q}, v) \leq (1+\epsilon)\ell(q^*, v)$. [6.1] Let $(\tilde{q}, v)$ be the last connection added to $C(Q', v)$. Let (q^*, v) be the next connection of $D(Q, v)$ that has not been considered yet. The algorithm adds (q^*, v) unless $\tilde{q}$ already semi ϵ_1-covers (q^*, v). The algorithm returns $C(Q, v) = C(Q_0, v) \cup C(Q_1, v)$.

We first prove that $C(Q, v)$ is a $(2\epsilon_0 + \epsilon_1)$-cover. Let q be a vertex in Q. Let d be the connection in $D(Q, v)$ which ϵ_0-covers q. Let c be a connection of $C(Q, v)$ that semi ϵ_1-covers d (it might be that $c = d$). We know that

$$\delta(q, c) \leq \delta(q, d) + \delta_Q(d, c) \text{ (triangle inequality)} \tag{6.2}$$

$$\delta_Q(d, c) + \ell(c, v) \leq (1+\epsilon_1)\ell(d, v) \text{ (c semi ϵ_1-covers d)} \tag{6.3}$$

$$\delta(q, d) + \ell(d, v) \leq (1+\epsilon_0)\delta(q, v) \text{ (d ϵ_0-covers q)} \tag{6.4}$$

We have that: $\delta(q, c) + \ell(c, v) \underset{(6.2)}{\leq} \delta(q, d) + \delta_Q(d, c) + \ell(c, v) \underset{(6.3)}{\leq} \delta(q, d) + (1 + \epsilon_1)\ell(d, v) \leq (1+\epsilon_1)(\delta(q, d) + \ell(d, v)) \underset{(6.4)}{\leq} (1+\epsilon_1)(1+\epsilon_0)\delta(q, v) = (1 + \epsilon_0 + \epsilon_1 + \epsilon_0\epsilon_1)\delta(q, v) \underset{\epsilon_1 \leq 1}{\leq} (1 + (2\epsilon_0 + \epsilon_1))\delta(q, v)$, and the approximation bound follows.

We now turn to show the generated cover is of $O(\epsilon_1^{-1})$ size. For $Q' \in \{Q_0, Q_1\}$, we show it is of size $O(\epsilon_1^{-1})$. Let $\{c_i\}_{i \geq 1}$, of size k, be the chosen connections along Q', numbered by their order along Q' toward the other endpoint t of Q', starting with $c_1 = \bar{q}$. We examine the function $f(c_i) = \delta_Q(t, c_i) + \ell(c_i, v)$. We observe that $f(c_i) - f(c_{i+1}) = (\delta_Q(t, c_i) + \ell(c_i, v)) - (\delta_Q(t, c_{i+1}) + \ell(c_{i+1}, v)) = \ell(c_i, v) + \delta_Q(c_i, c_{i+1}) - \ell(c_{i+1}, v) \geq \epsilon_1\ell(c_{i+1}, v) \geq \epsilon_1\ell(\bar{q}, v) \geq \epsilon_1\delta(\bar{q}, v)$. Thereby, $f(c_{i+1}) \leq f(c_1) - i\epsilon_1\delta(\bar{q}, v)$, hence $f(c_k) \leq f(c_1) - (k-1)\epsilon_1\delta(\bar{q}, v)$. Note that $f(c_k) = \delta_Q(t, c_k) + \ell(c_k, v) \geq \delta(t, v) \geq \delta_Q(t, \bar{q}) - \delta_Q(\bar{q}, v)$. Using the lower and upper bounds over $f(c_k)$, we have that $\delta_Q(t, \bar{q}) - \delta(\bar{q}, v) \leq f(c_k) \leq f(c_1) - (k-1)\epsilon_1\delta(\bar{q}, v) = \delta_Q(t, \bar{q}) + \ell(\bar{q}, v) - (k-1)\epsilon_1\delta(\bar{q}, v) \leq \delta_Q(t, \bar{q}) + (1+\epsilon_0)\delta(\bar{q}, v) - (k-1)\epsilon_1\delta(\bar{q}, v)$. Hence $(k-1)\epsilon_1 - (1+\epsilon_0)\delta(\bar{q}, v) \leq \delta_Q(\bar{q}, v)$ and so $k \leq 1 + \frac{2+\epsilon_0}{\epsilon_1}$. Therefore the size of the connections obtained over both $\{Q_0, Q_1\}$ is $O(\epsilon_1^{-1})$.

To obtain an ϵ-cover for a label we need the following *extended thinning procedure*, which converts a set of ϵ-covers, one for each λ-labeled vertex, into a single small ϵ-cover of label λ.

Lemma 6.3. *Let $\{u_i\}$ be vertices and Q be a shortest path in an undirected graph G. Given ordered ϵ-covering sets $\{D(Q, u_i)\}$ it is possible to compute in linear time a clean and ordered 3ϵ-covering connections set C of size $O(\epsilon^{-1})$ which represent approximated distances from any $q \in Q$ to its closest vertex among $\{u_i\}$.*

[6.1] The semi ϵ-cover definition is similar to ϵ-cover definition. The only difference is that $\delta(q^*, v)$ was replaced by $\ell(q^*, v)$ for fast computation purposes.

Proof. We first convert every connection length $\ell(q, u_i)$ to reflect an approximated length from q to its closest vertex $u^* \in \{u_j\}$, rather than to u_i. We obtain these lengths using the fact that q is ϵ-covered with respect to u^* by some connection in $D(Q, u^*)$. Let Z_u be the graph composed of the following. See fig. 6.2 for an illustration.

1. $\bar{Q}$, the reduced form of Q to connections of all $\{D(Q, u_i)\}$.
2. vertices $\{u_i\}$, along with edges between each u_i to its connections, with lengths equal to the corresponding connection lengths.
3. vertex u, connected with zero-length edges to all $\{u_i\}$.

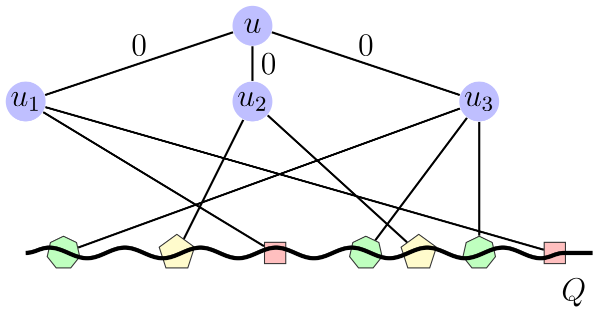

Fig. 6.2: Illustration of the situation in the proof of the extended thinning lemma (lemma 6.3). Each vertex in $\{u_i\}$ has different connections on Q. The distance from u_1 to any connection of u_2 is approximated using the connections of u_1.

By the ϵ-covering property, the distances between every $q \in \bar{Q}$ and u in Z_u represent approximate distances between q and its closest vertex $u^* \in \{u_j\}$ in G. To see this, assume q is a connection of u_1, closest to u^*. Let q^* be a connection of u^* which ϵ-covers q w.r.t. u^*. Then $\delta_{Z_u}(q, u^*) \leq \delta_Q(q, q^*) + \ell(q^*, u^*) = \delta_G(q, q^*) + \ell(q^*, u^*) \leq (1 + \epsilon)\delta_G(q, u^*)$.

It is possible to compute all shortest paths from u in Z_u in linear time; first, relax all edges incident to u and $\{u_i\}$. Then, relax the edges of $\bar{Q}$ by going first in one direction along Q and then relaxing the same edges again in the other direction. For connection p on $\bar{Q}$, a u-to-p shortest path first reaches Q along one of $\{u_i\}$ edges and then walks along Q toward p. Hence the relaxation was done in the correct order. We update the connection lengths to the distances thus computed.

Let $\tilde{D}(Q, u)$ denote the ordered union of all connections, along with the updated connection lengths. Since all $\{D(Q, u_i)\}$ were ordered, it is possible to order their union in linear time. Let G_u be the graph obtained from G by adding an apex u connected with zero length edges to all $\{u_i\}$. We stress that G_u is not constructed by the algorithm, but only used in the proof. $\tilde{D}(Q, u)$ is an ϵ-cover of Q with respect to u in G_u. Now apply lemma 6.2 to $\tilde{D}(Q, v)$ with $\epsilon_0 = \epsilon_1 = \epsilon$ to obtain a 3ϵ-cover of Q with respect to u in G_u of size $O(\epsilon^{-1})$.

6.2 Oracle for undirected graphs

As in the directed case, let $\mathcal{L}_r$ denote the set of labels in G_r°. The data structure kept and query algorithm remain as in the directed case. It remains to show how the connections are computed. We begin with the type-0 connections. For every $r \in \mathcal{T}$, for every $Q \in S_r \cup F_r$, the algorithm computes ordered ϵ-covering sets of connections on Q w.r.t. each vertex of G_r° by invoking lemma 4.8 to G_r°. This takes $O(\epsilon^{-1}|V(G_r^\circ)|\lg n)$ time (using [HKRS97] for shortest path computation). For each $\lambda \in \mathcal{L}_r$, let n_λ denote the number of λ-labeled vertices in G_r°. The total number of connections to λ-labeled vertices in G_r° is $O(\epsilon^{-1}n_\lambda)$. The algorithm next applies the extended thinning lemma (lemma 6.3) to get a connections set $C(Q,\lambda)$ of size $O(\epsilon^{-1})$ in $O(\epsilon^{-1}n_\lambda)$ time. Since $\sum_\lambda n_\lambda = O(|V(G_r^\circ)|)$, the runtime for a single r and Q is $O(\epsilon^{-1}|V(G_r^\circ)|)$.

We now show how to compute the type-1 connections without invoking lemma 4.8 to the entire input graph G at every call.

Lemma 6.4. *Let $r \in \mathcal{T}$. Type-1 connections of r to label $\lambda \in \mathcal{L}_r$ can be computed using just the (type-0) connections of strict ancestors of r. Computing all type-1 connections to $\mathcal{L}_r$ for all $r \in \mathcal{T}$ can be done in $O(\epsilon^{-2}n\lg^3 n)$.*

Proof. Recall the definition of the graph $\overrightarrow{X_r^Q}$ in lemma 5.1. In $\overrightarrow{X_r^Q}$, every artificial vertex $\lambda \in \mathcal{L}_r$ has arcs representing connections from paths on separators of ancestors of r.

Let $\overleftrightarrow{X_r^Q}$ be the graph identical to $\overrightarrow{X_r^Q}$, except its edges are undirected. The algorithm breaks every artificial vertex λ in $\overleftrightarrow{X_r^Q}$ into many copies $\{\lambda_e\}$, one per incident edge of λ. We stress that the artificial vertices λ_e are not directly connected to each other in $\overleftrightarrow{X_r^Q}$. Hence, the problem of shortcuts mentioned earlier is avoided. See fig. 6.3 for an illustration.

Note that splitting vertices in this way does not increase the number of edges in the $\overleftrightarrow{X_r^Q}$. The algorithm applies lemma 4.8 to $\overleftrightarrow{X_r^Q}$ and Q, obtaining a small sized ϵ-cover $C(Q,\lambda_e)$ for every λ_e.

Let q be any vertex of $\bar{Q}$, and let λ be a label in G_r°. Let P be a shortest q-to-λ path in G. Let r' be the rootmost strict ancestor of r such that $S_{r'}$ is intersected by P. Note that r' must exist since $q \in F_r$, so q belongs to the separator of some strict ancestor of r. Thus P is entirely contained in $G_{r'}^\circ$. Let Q' be a path in $S_{r'}$ intersected by P. By construction of $\overleftrightarrow{X_r^Q}$, it contains an ϵ-covering set of connections of Q' with respect to q in $G_{r'}^\circ$, as well as the edges of $\bar{Q}'$ and an ϵ-covering set of connections of Q' with respect to λ in $G_{r'}^\circ$. Hence, by lemma 6.1, there exists a shortest q-to-λ_e path (for some artificial vertex λ_e) in $\overleftrightarrow{X_r^Q}$ whose length is at most $(1+\epsilon)$ times the length of P. On the other hand, because the

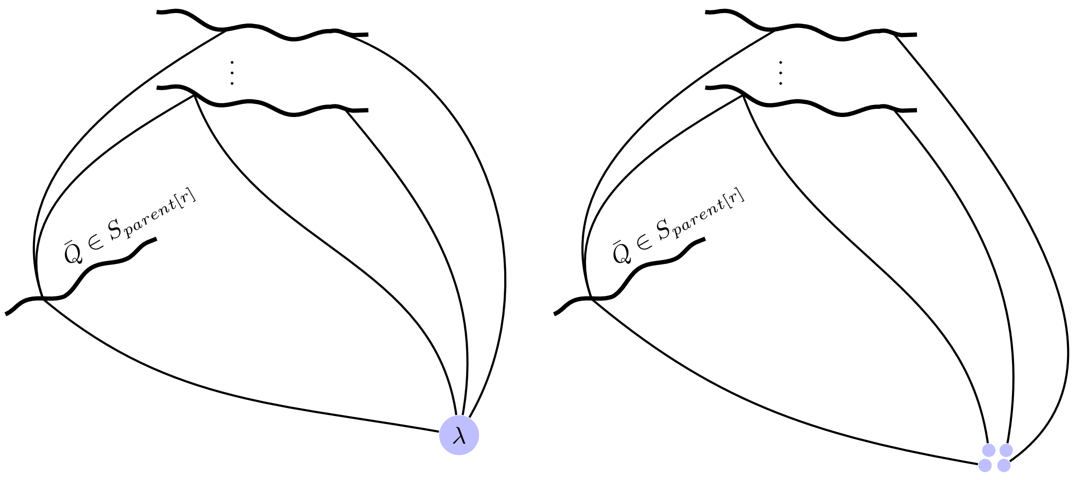

Fig. 6.3: Illustration of the utility of splitting an artificial vertex λ. On the left undesired shortcuts (teleportation) might occur. On the right ($\overleftrightarrow{X_r^Q}$) teleportation does not occur.

vertices λ_e (for any $\lambda \in \mathcal{L}_r$) are not directly connected to each other in $\overleftrightarrow{X_r^Q}$, every path in $\overleftrightarrow{X_r^Q}$ corresponds to some path in G, so shortest paths in $\overleftrightarrow{X_r^Q}$ are at least as long as those in G. This proves that $\overleftrightarrow{X_r^Q}$ correctly represents all desired type-1 connection lengths.

We proceed with describing the construction of the connection sets of the appropriate sizes. To bound the size of the connections $\{C(Q, \lambda_e)\}$, we count the number of edges incident to λ in X_r^Q (i.e., before it is split). There is an edge for each of the $O(\epsilon^{-1})$ connections of λ on each of the $O(\lg n)$ paths of separators of ancestors of r. For each such edge there is a vertex λ_e with an ϵ-covering set of $\bar{Q}$ of size $O(\epsilon^{-1})$. Thus, the total number of connections of $\bar{Q}$ for all λ_e vertices is $O(\epsilon^{-2} \lg n)$. The algorithm applies lemma 6.3, the extended thinning procedure, to $\{C(Q, \lambda_e)\}_e$ to get $C(Q, \lambda)$ of size $O(\epsilon^{-1})$. Doing so for all labels in G_r° requires $O(\epsilon^{-2} \lg n + \epsilon^{-1}|\mathcal{L}_r|)$ space.

We now bound the running time. Since splitting vertices does not increase the number of edges, applying lemma 4.8 to $\overleftrightarrow{X_r^Q}$ takes $O(\epsilon^{-2}|V(G_r^\circ)| \lg^2 n)$ time. Applying lemma 6.3 is done within the same time bound. To conclude, the total runtime over all nodes of $\mathcal{T}$ is $O(\epsilon^{-2} n \lg^3 n)$.

We have thus established:

Theorem 6.1. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\lg\lg n + \epsilon^{-1})_{time}\rangle$ Vertex-Label Distance Oracle can be constructed within $O(\epsilon^{-2} n \lg^3 n)$ time w.h.p.* [6.2] *in an undirected planar graph with n vertices.*

6.3 Faster constructions

In order to achieve faster construction, one may avoid computing type-1 connections. When queried for a u-to-λ distance, one may now examine each ancestor r' of r_u to figure the u-to-λ distance in $G^{\circ}_{r'}$. Since a shortest path is confined to $G^{\circ}_{r'}$ for some ancestor r' of r_u (and crosses its separator $S_{r'}$), this procedure results in the correct distance approximation. Avoiding the construction of type-1 connections decreases the construction time by a factor of $O(\epsilon^{-1}\lg n)$ in both the directed and undirected cases (using [HKRS97] for shortest path computations). We thus obtain the following:

Theorem 6.2. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}\ ,\ O(\epsilon^{-1}\lg n \lg\lg(nN))_{time}\rangle$ Vertex-Label Distance Oracle can be constructed within $O(\epsilon^{-1} n \lg^2 n \lg(nN))$ time w.h.p.* [6.3]

Theorem 6.3. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\epsilon^{-1}\lg n)_{time}\rangle$ Vertex-Label Distance Oracle can be constructed within $O(\epsilon^{-2} n \lg^3 n)$ time w.h.p.*

Comparison to [LMN13] We note that the oracle of Li, Ma, and Ning [LMN13] is similar to the one in Theorem 6.3. One difference is that they use an algorithm of Klein [Kle05] to construct the covering sets. Klein's algorithm only works for planar graphs, and therefore cannot be used to compute type-1 connections using the non-planar X^Q_r, as we do. Another difference is that [LMN13] compute a small set of connections for each λ-labeled vertex, whereas we further combine these sets into a single small set for the label λ using the thinning procedure (lemma 6.3). This affects the query time; [LMN13] need to perform a predecessor search in the set of connections that may be as large as n, whereas we only work with a set of connections of size $O(\epsilon^{-1})$. In [LMN13] a $O(\log n)$-time predecessor search was used, resulting in a $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\epsilon^{-1}\lg^2 n)_{time}\rangle$ vertex-label oracle.

[6.2] The probability in the construction time is only due to the use of perfect hashing.

[6.3] The construction described in the directed case cannot use [HKRS97] linear time shortest path algorithm. This is because we deal with non-planar graphs. However, using the similar splitting technic as in the undirected case, such algorithm can be formed.

7 Reporting Approximate Shortest Path

Thorup describes how to augment his oracle to report a u-to-v path of length $(1+\epsilon)\delta(u,v)$ in time linear in the number of edges reported. We provide here a brief description (cf. [Tho04, Sections 2.7, 2.8, 3.7]) for the undirected case (the directed case is similar).

The algorithm now stores additional information. The ϵ-cover construction algorithm (lemma 4.4) computes shortest path trees rooted at each connection. In the original description these trees are discarded once the connection lengths have been recorded. To report shortest paths, the algorithm stores these trees for all type-0 connections. Let r be a node of $\mathcal{T}$. For each $v \in G_r^\circ$, $Q \in S_r \cup F_r$ and type-1 connection $q \in C(Q,v)$, (i.e., connections computed by invoking the ϵ-cover construction algorithm on X_r^Q), the algorithm records the rootmost node $r' \in \mathcal{T}$ whose separator is intersected by the q-to-v shortest path in X_r^Q.

The query algorithm for the distance between u and v uses some type-1 connection. Let r' be the node of $\mathcal{T}$ recorded for that connection. By choice of r' there exists a $(1+\epsilon)$-approximate shortest path P between u and v in $G_{r'}^\circ$. The algorithm now uses the type-0 connections of r' to find P and uses the shortest path trees stored for those connections to report the edges of P.

Storing all shortest path trees does not change the preprocessing time but increases the required space by $O(\epsilon^{-1} n \log^2 n)$. This is because each vertex participates in $O(\epsilon^{-1}\log n)$ connections shortest path trees (see lemma C.4) in $O(\log n)$ nodes of $\mathcal{T}$. Let $\bar{d}$ be the number of edges of the reported path. The resulting query time is $O(\epsilon^{-1} + \log\log n + \bar{d})$.

We extend this technique to the vertex label case. As in the vertex-vertex case, the query algorithm finds a node r' such that there exists in $G_{r'}^\circ$ a path P that $(1+\epsilon)$-approximates the shortest path between u and a λ-labeled vertex in G. It then finds P using the type-0 connections of r'. The main difference is that at this point the algorithm knows the distance to the artificial vertex λ, but not the identity of the λ-labeled vertex realizing this distance. However, this information can be stored along with the shortest path trees for type-0 connections. In the directed case this is done by storing the vertices preceding the artificial vertices in the shortest path trees computed during the ϵ-cover construction algorithm. In the undirected case this can be done during the extended thinning procedure, which converts distances to labeled vertices into distances to artificial vertices.

8 Concluding Remarks

This work presents an extension of Thorup's vertex-to-vertex oracle to enable vertex-to-label queries in planar graphs. Although our focus is on planar graphs, the algorithm works for any class of graphs which are both minor-closed and tree-path separable (replacing [HKRS97] shortest paths algorithm with a simple dijkstra algorithm). In the directed case, the extension can be done by "injecting" apices to the graph, thereby augmenting the data-structure for that purpose. However, as argued, this technic does not work in the undirected case. In the undirected case, we have shown how to "manually" generate the missing data in order to enable vertex-label queries. The query time of our data structure is nearly constant, and faster by a factor of $\frac{\epsilon^{-1}\lg n \lg \rho}{\epsilon^{-1}+\lg\lg n}$ compared to the previous best result [LMN13], using the same space and nearly the same preprocessing time.

Related problems In this work we dealt with labeled-graphs where each vertex has exactly one label. However, there is no obstacle to deal with labeled-graphs in which each vertex has several labels. Let κ be the bound on the maximal number of labels a vertex is labeled by. The preprocessing time and space of the oracles in this work are multiplied by κ.

We note that similar technics to the ones presented in this work enable constructing oracles for apex graphs (i.e., planar graphs with additional apices). This is because our technics are based on computing an ϵ-cover set for an apex which represents a label. The apices can be thought of as inducing labels on the planar part of the graph. Note, however, that when apices are real vertices in the graph (as opposed to artificial vertices), type-0 ϵ-covering sets for apices can be computed directly using lemma 4.4 or lemma 4.8. Type-1 ϵ-covering sets are produced in the same manner as presented in this work. The preprocessing time and space bounds increase according to the value of κ induced by the apices.

A possible direction for future work is to devise efficient label-to-label distance queries. This seems significantly more difficult. In the vertex-to-label query, when queried for a u-to-λ distance the algorithm used the leafmost node r in $\mathcal{T}$ that contains u and some λ-labeled vertex to quickly answer the query. In a λ_1-to-λ_2 query, we do not know which node $r \in \mathcal{T}$ necessarily has its separator intersected by the desired shortest path. This is because λ_1 and λ_2 labeled vertices might be scattered in any of the leaves of $\mathcal{T}$. Constructing a label-to-label distance oracle whose query time does not depend on the number of vertices with the queried labels remains as an interesting open problem.

References

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

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

[BS06] Surender Baswana and Sandeep Sen, *Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time*, ACM Transactions on Algorithms **2** (2006), no. 4, 557–577.

[Cab12] Sergio Cabello, *Many distances in planar graphs*, Algorithmica **62** (2012), no. 1-2, 361–381.

[Che12] Shiri Chechik, *Improved distance oracles and spanners for vertex-labeled graphs*, Proceedings of the 20th European Symposium on Algorithms (ESA), 2012, pp. 325–336.

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

[Dji96] Hristo Djidjev, *On-line algorithms for shortest path problems on planar digraphs*, Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG), 1996, pp. 151–165.

[FR06] Jittat Fakcharoenphol and Satish Rao, *Planar graphs, negative weight edges, shortest paths, and near linear time*, Journal of Computer and System Sciences **72** (2006), no. 5, 868–889.

[HKRS97] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian, *Faster shortest-path algorithms for planar graphs*, Journal of Computer and System Sciences **55** (1997), no. 1, 3–23.

[HLWY11] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster, *Distance oracles for vertex-labeled graphs*, Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP), 2011, pp. 490–501.

[HT84] Dov Harel and Robert Endre Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM Journal on Computing **13** (1984), no. 2, 338–355.

[KKS11] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer, *Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs*, Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP), 2011, pp. 135–146.

[Kle02] Philip N. Klein, *Preprocessing an undirected planar network to enable fast approximate distance queries*, Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 820–827.

[Kle05] ______, *Multiple-source shortest paths in planar graphs*, Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 146–155.

[KST13] Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup, *More compact oracles for approximate distances in undirected planar graphs*, Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2013, pp. 550–563.

[LMN13] Mingfei Li, Chu Chung Christopher Ma, and Li Ning, $(1+\epsilon)$*-distance oracles for vertex-labeled planar graphs*, Proceedings of the 10th International Conference on Theory and Applications of Models of Computation, 10th International Conference (TAMC), 2013, pp. 42–51.

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

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

[MW10] Shay Mozes and Christian Wulff-Nilsen, *Shortest paths in planar graphs with real lengths in* $O(n \log^2 n/ \log\log n)$ *time*, Proceedings of the 18th European Symposium on Algorithms (ESA), 2010, pp. 206–217.

[Nus11] Yahav Nussbaum, *Improved distance queries in planar graphs*, Proceedings of the 14th International Workshop on Algorithms and Data Structures (WADS), 2011, pp. 642–653.

[Tho04] Mikkel Thorup, *Compact oracles for reachability and approximate distances in planar digraphs*, Journal of the ACM **51** (2004), no. 6, 993–1024.

[TPSS11] Yufei Tao, Stavros Papadopoulos, Cheng Sheng, and Kostas Stefanidis, *Nearest keyword search in XML documents*, Proceedings of the International Conference on Management of Data (SIGMOD), 2011, pp. 589–600.

[TZ05] Mikkel Thorup and Uri Zwick, *Approximate distance oracles*, Journal of the ACM **52** (2005), no. 1, 1–24.

[Wul12] Christian Wulff-Nilsen, *Approximate distance oracles with improved preprocessing time*, Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2012, pp. 202–208.

[Wul13] ______, *Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time*, Computational Geometry **46** (2013), no. 7, 831–838.

Appendix

A Thorup's Reductions [Tho04]

We present two main reductions [Tho04] uses. The first reduction (reduction 1) enables keeping linear space so that we may:

- Assume a threshold $O(\alpha)$ over the diameter of a reduced form of G
- Obtain a (t, α)-layered spanning tree of a reduced form of G

The second reduction enables to compose the final approximate distance oracle given a scale-(α, ϵ) distance oracle. Since it assumes a constant α, it encapsulates reduction 1. We will get into more details in reduction 2.

Reduction 1 *(*Length reduction*) Let G be a graph and $\alpha \in \mathbb{R}^+$. We can construct a set of graphs $\{G_i^\alpha\}_{i=0\ldots k-1}$ in linear time, which has the following properties:*

1. *$\sum |G_i^\alpha| = O(|G|)$, where $|G| = |V| + |E|$.*
2. *Each $v \in V$ has an index $\iota(v)$ s.t. any $w \in V$ has $d = \delta_G(v, w) \le \alpha$ iff $d = \delta_{G^\alpha_{\iota(v)-2}}(v, w) = \delta_{G^\alpha_{\iota(v)-1}}(v, w) = \delta_{G^\alpha_{\iota(v)}}(v, w)$*
3. *Each G_i^α is a $(3, \alpha)$-layered graph.*
4. *Each G_i^α is a minor of G. Meaning it can be achieved by contraction and deletion of arcs and vertices. Note that when G is planar, so does G_i^α.*

The reader is referred to [Tho04, lemma 2.2, 3.1] for full proof.
We present the scale reduction. [A.1]

Reduction 2 *(*Scale reduction*) Let $\mathcal{A}_{(\alpha,\epsilon')}(\cdot)$ be an algorithm that outputs a scale-(α, ϵ') $\langle O(s(n, \epsilon'))_{space}\, , O(t(\epsilon'))_{time}\rangle$ distance oracle. Then there is an algorithm which outputs $(1 + \epsilon)$-stretch $\langle O(s(n, \epsilon) \lg(nN))_{space}\, , O(t(\frac{1}{4}) \lg\lg(nN) + t(\frac{\epsilon}{4}))_{time}\rangle$ distance oracle.*

The final distance oracle is constructed out of scale-(α, ϵ') distance oracles for $(\alpha, \epsilon') \in \{2^i\}_{i\in[\lceil\lg(nN)\rceil]} \times \left\{\frac{1}{4}, \frac{\epsilon}{4}\right\}$. Denote $\tilde{\delta}^{(\alpha,\epsilon')}$ the query result of scale-(α, ϵ') algorithm. Given (u, w) the algorithm first finds quickly a value of α s.t. $\frac{\alpha}{2} \le \tilde{\delta}^{(\alpha,\frac{1}{4})}(u, w) \le \alpha$ using binary search over α. For that α we know that $\frac{\alpha}{4} \le \delta(u, w)$. Hence $\tilde{\delta}^{(\alpha,\frac{\epsilon}{4})}(u, w)$ is the wanted result, since $\frac{\tilde{\delta}^{(\alpha,\frac{\epsilon}{4})}(u,w)}{\delta(u,w)} \le \frac{\delta(u,w)+\epsilon\frac{\alpha}{4}}{\delta(u,w)} \le 1 + \epsilon$.
We mention that the original proof can be found in [Tho04, lemma 3.8].

[A.1] It appears to us as inaccurate, concerning constants, as presented in [Tho04].

B Thorup's Recursion [Tho04]

Given a frame F and a subgraph H, we describe our goals in the recursion obtaining the recursive graph decomposition $\mathcal{T}$; first, keep minimal number of relevant frame paths in each recursion step (which lowers the query time). Second, keep $|F|$ of $O(|H|)$ size in all recursive calls (which lowers the preprocessing time).

Maintaining Few Frame paths To do so the algorithm applies lemma 4.2 twice:

1. Giving all arcs of H unit weights, and zero to all others.
2. Giving all F's leaves arcs unit weights, and zero to all others.

The number of paths in F is constant since the first increases the number of F branches by 2, while the second divides the number of F's branches by half in each preceding step.

Maintaining Small Subgraph Size In order to ensure $|F| = O(|H|)$, the algorithm reduces F into F's vertices which belong to H, or are branching points of F. Note that since there are constant number of frames, there are only a constant number of branching points. This can be done in $O(|F| + |H|)$ time. See [Tho04, section 2.5.2] for full details.

C Thorup's ϵ-covering Sets [Tho04]

[Tho04] shows how to achieve connections from/to a shortest path efficiently. We outline the process which achieve these connections and their lengths. For the following notations we leave out specific definitions for $C(Q, v)$ as they are inferred. We consider a shortest path Q in a $(3, \alpha)$-layered graph G. We denote the connection length of a pair q w.r.t. vertex v by $\ell(v, q)$.

– A connection q ϵ-covers q^* w.r.t. vertex v if q precedes q^* on Q and

$$\ell(v, q) + \delta(q, q^*) \leq \delta(v, q^*) + \epsilon\alpha \tag{C.1}$$

– A connections set $C \subseteq V(Q)$ is called *ϵ-covering* w.r.t. vertex v if for every $q \in Q$ there is a connection q^* in C that ϵ-covers it w.r.t. v.
– We denote ϵ-covering sets $\{C(v, Q)\}_{v \in G}$ by $C(H, Q)$.

The following is a refinement of the statement of lemma 4.3.

Lemma C.1. *Assume a shortest u-to-w path P in G crosses a shortest path Q. Let $C(u, Q), C(Q, w)$ be ϵ-covering sets. Let G_Q^{uw} be the graph with vertices u, w, the vertices of the reduction of Q to $C(u, Q) \cup C(Q, w)$, and with u-to-Q and Q-to-v arcs whose lengths are the corresponding connections lengths of $C(u, Q)$ and $C(w, Q)$.*

$$\delta_{G_Q^{uw}}(u, w) \leq \delta_G(u, w) + 2\epsilon\alpha \tag{C.2}$$

Proof. Assume q^* is the first vertex of Q that P crosses. Let q_0, q_1 denote the connections which ϵ-cover q^* in $C(u, Q), C(Q, w)$ w.r.t. u, w respectively. The path $u \rightsquigarrow q_0 \rightsquigarrow q_1 \rightsquigarrow w$ in G_Q^{uw} is of length $\ell(u, q_0) + \delta_Q(q_0, q_1) + \ell(q_1, w) = \ell(u, q_0) + \delta_G(q_0, q^*) + \delta_G(q^*, q_1) + \ell(q_1, w) \leq \delta_G(u, q^*) + \epsilon\alpha + \delta_G(q^*, w) + \epsilon\alpha = \delta_G(u, w) + 2\epsilon\alpha$. [C.1] The original proof can be found in [Tho04, lemma 3.5].

We now outline how $C(H, Q) \cup C(Q, H)$ can be used to perform a quick distance queries.

Definition C.1. *$C(v, Q)$ is called* ordered *if it is sorted by the distances of the connections along Q from Q's beginning.*

[C.1] Assume G is a subgraph and Q a part of it's infinite face. Note that in order to approximate the u-to-w distance in the entire graph, it suffices to keep connection lengths of $C(u, Q)$ which respect distances strictly in G and connection lengths $C(Q, v)$ in the entire graph.

Definition C.2. *An ϵ-covering $C(v, Q)$ is called* clean *if it has no strict subset which is also ϵ-covering. In other words, the connections are spread along Q as much as possible and there is no connection redundancy. Formally, $\forall v \in H\ \forall a \in C(v, Q) \nexists b \in C(v, Q).\ \ell(v, a) + \delta_Q(a, b) \leq \ell(v, b)$.*

We can now present [Tho04, lemma 3.11] which later on will be used to thin covers sizes:

Lemma C.2. *Let $D(Q, v)$ be an ordered ϵ_0-covering set. A clean and ordered $(\epsilon_0 + \epsilon_1)$-covering set $C(Q, v) \subseteq D(Q, v)$ of size $O(\epsilon_1^{-1})$ can be constructed in $O(|D(Q, v)|)$ time.*

Proof. This is done by simply enumerating $D(Q, v)$ and discarding unnecessary connections. Again, the reader is referred to the original lemma for full details.

Lemma C.3. *Given clean and ordered $C(u, Q)$ and $C(Q, w)$, the distance from u to w through Q can be approximated in $O(|C(u, Q)| + |C(Q, w)|)$ time.*

Proof. The correctness follows from lemma C.1. The approximation is done by first merging the two sets by their connection's order. The merge is done in linear time because the sets are already ordered. Second, both sets are clean and hence u's and w's connections which achieve minimal length (see lemma C.1) must be consecutive in the merged list. This is done in linear time, as claimed in [Tho04, lemma 3.6].

We now outline how $C(Q, H)$ sets can be achieved efficiently for a given shortest path $Q \subset H$. Like [Tho04], we will leave out the details concerning $C(H, Q)$ throughout [Tho04] outline explanation. These details can be induced by changing the direction of the arcs and applying the same technic.

We outline the proof of [Tho04, lemma 3.12] which is among the core lemmas.

Lemma C.4. *Let H be a $(3, \alpha)$-layered graph and Q a shortest path. ϵ-covering sets $C(Q, H)$ can be constructed in $O(\epsilon^{-1} sssp(Q, H)|E(H)| \lg |V(Q)|)$ time, each of size $O(\epsilon^{-1} \lg n)$.*

Proof. The algorithm starts by first letting $Q_0 = Q$ and $H_0 = H$. Denote a (c) the start (end) of Q_0. The algorithm assumes that:

1. Q_0 is reduced to vertices belong to H_0 - for otherwise this can be done in $O(|V(Q_0)|)$ time.
2. It has shortest paths distances from the end points of Q_0 to H_0 - for otherwise it is computed them in $O(sssp(H, Q)|E(H_0)|)$ time.
3. It already inserted $\delta_{H_0}(a, v)$ and $\delta_{H_0}(c, v)$ as connections - for otherwise the algorithm do so in $O(|E(H_0)|)$.

The algorithm now takes the unweighted middle vertex b of Q and calculates another shortest paths from it. Denote Q_1 (Q_2) the part of Q that is before (after) b. For $Q^{'} \in \{Q_1, Q_2\}$, let s (t) be the start (end) of $Q^{'}$. We present a new definition, (q, v) *semi-ϵ-covers* (q^*, v) if q precedes q^* on $Q^{'}$ and $\delta(q, q^*) + \ell(q^*, v) \leq \ell(q, v) + \epsilon\alpha$. Denote $U = \{v | \ (s, v)$ semi-ϵ-covers $\ (t, v)\}$, or it's distance from s is larger then 2α. The algorithm may now recurse over $(Q^{'}, H_0 \setminus U)$.

Thorup shows that when the algorithm reaches a recursion level where all intervals of Q are of length $\leq \epsilon\alpha$, each $v \in H$ must be covered in each of them. For full details of correctness and efficiency the reader is referred to [Tho04, lemma 3.12].

We mention that $C(Q, H)$ obtained by lemma C.4 are ordered. These sets are thinned in linear time as to enable small space and query time.

We capture these results by the following lemma:

Lemma C.5. *Given a graph $(3, \alpha)$-layered graph H and shortest path Q, $C(Q, H) \cup C(H, Q)$ sets of size $O(\epsilon^{-1})$ can be constructed in $O(\epsilon^{-1} sssp(Q, H)|E(H)| \lg |V(Q)|)$ time.*

Proof. Denote $\epsilon^{'} = \epsilon/2$. The algorithm uses lemma C.4 to compute ordered $C(Q, H) \cup C(H, Q)$ where each ϵ-covering set is of size $O(\epsilon^{'-1} \lg n)$. The algorithm thins the size of each set using lemma C.2 with $\epsilon_0 = \epsilon_1 = \epsilon^{'} = \epsilon/2$. By lemma C.3 a query can be performed in $O(\epsilon^{-1})$ time.

Combining lemma C.3 and lemma C.5, lemma 4.4 follows.

D Thorup's treatment of the undirected case [Tho04]

For the undirected case, q^* *ϵ-covers* q w.r.t. vertex v if $\delta(q, q^*) + \ell(q^*, v) \leq (1 + \epsilon)\delta(q, v)$.

We would have liked to convert the directed ϵ-cover construction procedure (lemma C.4) to the undirected case. However, the proof does not carry over. To deal with this difficulty we use an alternative definition of ϵ-cover [Tho04]: q^* *quasi ϵ-covers* q if $\delta(q, q^*) + \ell(q^*, v) \leq \delta(q, v) + \epsilon\ell(q^*, v)$. In the undirected case, a connections set $C \subseteq V(Q)$ is called *quasi ϵ-covering* if for every $q \in Q$ there is a connection q^* in C that quasi ϵ-covers it.

For the quasi ϵ-cover definition, the proof of the analogue of lemma C.4 (ϵ-covers construction) carries over to undirected graphs and produces quasi ϵ-covers each of size $O(\epsilon^{-1} \lg n)$.

Lemma D.1. *Given an undirected graph H and a shortest path Q, ϵ-covering sets of Q with respect to all vertices of H, each of size $O(\epsilon^{-1} \lg n)$, can be constructed in $O(\epsilon^{-1} sssp(Q, H)|E(H)| \lg(|V(Q)|))$ time.*

We show in proposition 1 below that quasi ϵ-cover is a 2ϵ-cover for any $\epsilon < 1/2$. Therefore, the quasi ϵ-cover produced by lemma D.1 can be thinned into a $O(\epsilon)$-cover of size $O(\epsilon^{-1})$ using the thinning lemma for the undirected case (lemma 6.2). This $O(\epsilon)$-cover can be used to answer distance queries, as shown in lemma 6.1.

Proposition 1. *Let $C(v, Q)$ be a quasi ϵ-covering set. For any $0 < \epsilon \leq 1/2$, $C(v, Q)$ is a 2ϵ-covering set.*

Proof. If q^* quasi ϵ-covers q then $\ell(q^*, v) \leq \frac{1}{1-\epsilon}\delta(q, v) \leq 2\delta(q, v)$. Hence $\delta(q, q^*) + \ell(q^*, v) \leq \delta(q, v) + \epsilon\ell(q^*, v) \leq (1 + 2\epsilon)\delta(q, v)$. Therefore, if $C(v, Q)$ is a quasi ϵ-covering set, it is a 2ϵ-covering set.

A flaw in Thorup's treatment of the undirected case

There is another notion of covering, apart from ϵ-covering and quasi ϵ-covering [Tho04].

Definition D.1. q^* strictly ϵ-covers q *w.r.t.* v *if* $\delta(q, q^*) + \ell(q^*, v) \leq \delta(q, v) + \epsilon\delta(v, Q)$.

In [Tho04], Thorup uses quasi ϵ-covers and strict ϵ-covers, but does not use (plain) ϵ-covers.[D.1] Most of the discussion in [Tho04] is devoted to the directed case, in which the term $\epsilon\delta(v, Q)$ in the strict ϵ-cover definition is replaced by $\epsilon\alpha$ for some constant α. When treating the undirected case, Thorup claims that all lemmas, except for the efficient construction procedure, carry over from the directed case to the undirected case when the directed definition of ϵ-covering is replaced with strict ϵ-covering. The treatment of

[D.1] Thorup did not use the terms strict and quasi.

the efficient construction for the undirected case is more detailed, where a procedure for efficiently constructing quasi ϵ-covering sets is given (lemma D.1, [Tho04, Lemma 3.18]).

We believe that the treatment of the undirected case in [Tho04] suffers from two flaws. First, the proof of the ϵ-covers thinning does not seem to carry over from the directed case to the undirected case when using quasi ϵ-covers; recalling our thinning lemma proof lemma 6.2, using the quasi ϵ-cover definition instead, a bound to ensure quasi ϵ-covering cannot be established as it depends on a term that cannot be bounded. [D.2] Second, since the construction is of quasi ϵ-covers, whereas all other parts of the undirected oracle in [Tho04] assume strict ϵ-covers, the correctness of the entire oracle is not established.

Our algorithm does not use strict ϵ-covers at all. We use Thorup's efficient construction of quasi ϵ-covers, which, by proposition 1 is also a $O(\epsilon)$-cover, and prove that the thinning procedure and query algorithm work for ϵ-covers.

[D.2] More specifically, $\delta(q, c) + \ell(c, v)$ of lemma 6.2 can be bounded by a term that contains $\ell(d, v)$. In that case, the term $\ell(d, v)/\ell(c, v)$ is not universally bounded. Therefore, the bound cannot be used to justify the thinning.

תקציר

אנו עוסקים במציאת מרחק ממיקום על מפה אל עבר ספק שירות (דוגמת תחנת רכבת, בית חולים וכו'). מטרתנו היא להגדיר אלגוריתם יעיל לבעיה זו במקרה והמפה היא "רדודה" (ניתנת לייצוג על ידי גרף מישורי) וכל קודקוד מזוהה עם תווית (המייצגת את ספק השירות הרלוונטי). לכל $\epsilon > 0$ קבוע, אנו מראים כיצד ניתן לחשב אוב מרחקים עבור גרף מישורי ממושקל בעל תוויות קודקודים, כך שבהינתן קודקוד u ותווית λ, מחזיר את המרחק המינימלי בין u לקודקוד בעל תווית λ הקרוב אליו ביותר, בקירוב של $(1+\epsilon)$ המרחק. עבור גרף n־קודקודי מישורי לא־מכוון, זמן העיבוד הוא $O(\epsilon^{-2} n \lg^3 n)$, בעל דרישת מקום של $O(\epsilon^{-1} n \lg n)$ וזמן תשאול של $O(\lg\lg n + \epsilon^{-1})$. למקרה המכוון בו משקל קשת חסום על ידי N, זמן העיבוד הוא $O(\epsilon^{-2} n \lg^3 n \lg(nN))$, המקום הוא $O(\epsilon^{-1} n \lg n \lg(nN))$ וזמן התשאול הוא $O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})$.

המרכז הבינתחומי הרצליה
בית ספר אפי ארזי למדעי המחשב

אוב מרחקים מונחה סיווג־יעד עבור גרפים מישוריים

עבודת תזה המוגשת כמילוי חלק מהדרישות לקראת תואר מוסמך במסלול מחקרי במדעי המחשב

מוגש על ידי אייל סקופ בהנחיית ד"ר שי מוזס

מרץ **2015**