Theoretical Computer Science 761 (2019) 78–88

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

# Constant query time $(1+\epsilon)$-approximate distance oracle for planar graphs[☆]

Qian-Ping Gu[*], Gengchun Xu

*School of Computing Science, Simon Fraser University, Burnaby, BC V5A1S6, Canada*

ARTICLE INFO

*Article history:*
Received 9 June 2017
Received in revised form 22 August 2018
Accepted 28 August 2018
Available online 29 August 2018
Communicated by A. Marchetti-Spaccamela

*Keywords:*
Distance oracle
Planar graphs
Approximate algorithms
Graph decomposition

ABSTRACT

We give a $(1+\epsilon)$-approximate distance oracle with $O(1)$ query time for an undirected planar graph $G$ with $n$ vertices and non-negative edge lengths. For $\epsilon > 0$ and any two vertices $u$ and $v$ in $G$, our oracle gives a distance $\tilde{d}(u,v)$ with stretch $(1+\epsilon)$ in $O(1)$ time. The oracle has size $O(n \log n((\log n)/\epsilon + f(\epsilon)))$ and pre-processing time $O(n \log n((\log^3 n)/\epsilon^2 + f(\epsilon)))$, where $f(\epsilon) = 2^{O(1/\epsilon)}$. This is the first $(1+\epsilon)$-approximate distance oracle with $O(1)$ query time independent of $\epsilon$ and the size and pre-processing time nearly linear in $n$, and improves the query time $O(1/\epsilon)$ of previous $(1+\epsilon)$-approximate distance oracle with size nearly linear in $n$.

© 2018 Elsevier B.V. All rights reserved.

## 1. Introduction

Finding a distance between two vertices in a graph is a fundamental computational problem and has a wide range of applications. For this problem, there is a rich literature of algorithms. This problem can be solved by a single source shortest path algorithm such as the Dijkstra and Bellman–Ford algorithms. In many applications, it is required to compute the shortest path distance in an extremely short time. One approach to meet such a requirement is to use distance oracles.

A distance oracle is a data structure which keeps the pre-computed distance information and provides a distance between any given pair of vertices very efficiently. There are two phases in the distance oracle approach. The first phase is to compute the data structure for a given graph $G$ and the second is to provide an answer for a query on the distance between a pair of vertices in $G$. The efficiency of distance oracles is mainly measured by the time to answer a query (*query time*), the memory space required for the data structure (*oracle size*) and the time to create the data structure (*pre-processing time*). Typically, there is a trade-off between the query time and the oracle size. A simple approach to compute a distance oracle for graph $G$ of $n$ vertices is to solve the all pairs shortest paths problem in $G$ and keep the shortest distances in an $n \times n$ distance array. This gives an oracle with $O(1)$ query time and $O(n^2)$ size. A large number of papers have been published for distance oracles with better measures on the product of query time and oracle size, see Sommer's paper for a survey [20].

Planar graphs are an important model for many networks such as the road networks. Distance oracles for planar graphs have been extensively studied. Djidjev proves that for any oracle size $S \in [n, n^2]$, there is an exact distance oracle with query time $O(n^2/S)$ for weighted planar graphs [9]. There are exact distance oracles with size $O(S)$ and more efficient query time for different ranges of $S$, for example, an oracle by Wulff-Nilsen [23] with $O(1)$ query time and $O(n^2(\log\log n)^4/\log n)$ size

☆ A preliminary version of the paper appeared in the Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC 2015) [13].

* Corresponding author.
*E-mail addresses:* qgu@cs.sfu.ca (Q.-P. Gu), gxa2@sfu.ca (G. Xu).

https://doi.org/10.1016/j.tcs.2018.08.024
0304-3975/© 2018 Elsevier B.V. All rights reserved.

and an oracle by Cohen-Added et al. [7] with $O(\log n)$ query time and $O(n^{5/3})$ size for weighted directed planar graphs. Recently, Gawrychowski et al. [12] improve the result in [9] and show that for any oracle size $S \in [n, n^2]$, there is an exact distance oracle with query time $\tilde{O}\max\{1, (n^{1.5}/S)\}$ for weighted directed planar graphs. The oracle in [12] has query time $O(\log n)$ for $S \in [n^{1.5}, n^2]$, which improves the result in [7]. Readers may refer to Sommer's survey paper [20] for more information.

Approximate distance oracles have been developed to achieve very fast query time and near linear size for planar graphs. For vertices $u$ and $v$ in graph $G$, let $d_G(u,v)$ denote the distance between $u$ and $v$. An oracle is called an $\alpha$-approximate oracle or with *stretch* $\alpha$ for $\alpha \geq 1$ if it provides a distance $\tilde{d}(u,v)$ with $d_G(u,v) \leq \tilde{d}(u,v) \leq \alpha d_G(u,v)$ for $u$ and $v$ in $G$. An oracle is said to have an *additive stretch* $\beta \geq 0$ if it provides a distance $\tilde{d}(u,v)$ with $d_G(u,v) \leq \tilde{d}(u,v) \leq d_G(u,v) + \beta$. For $\epsilon > 0$, Thorup gives a $(1+\epsilon)$-approximate distance oracle with $O(1/\epsilon)$ (resp. $O(1/\epsilon + \log\log\Delta)$, where $\Delta$ is the longest finite distance between any pair of vertices in $G$) query time and $O(n\log n/\epsilon)$ (resp. $O(n(\log\Delta)\log n/\epsilon)$) size for an undirected (resp. directed) planar $G$ with non-negative edge lengths [21]. A similar result for undirected planar graphs is found independently by Klein [16]. Kawarabayashi et al. give a $(1+\epsilon)$-approximate distance oracle with $O((1/\epsilon)\log^2(1/\epsilon)\log\log(1/\epsilon)\log^* n)$ query time and $O(n\log n\log\log(1/\epsilon)\log^* n)$ size for undirected planar graphs with non-negative edge lengths [15]. The query times of the oracles above are fast but still at least $O(1/\epsilon)$. Recently, Wulff-Nilsen gives a $(1+\epsilon)$-approximate distance oracle with $O(n(\log\log n)^2/\epsilon + \log\log n/\epsilon^2))$ size and $O((\log\log n)^3/\epsilon^2 + \log\log n\sqrt{\log\log((\log\log n)/\epsilon^2)}/\epsilon^2)$ query time for undirected planar graph with non-negative edge lengths [24]. This result has a better trade-off between the query time and the oracle size in the size of graph than those in [15,16,21].

Distance oracles with constant query time are of both theoretical and practical importance [6,8]. Our main result is an $O(1)$ query time $(1+\epsilon)$-approximate distance oracle for undirected planar graphs with non-negative edge lengths.

**Theorem 1.** *Let $G$ be an undirected planar graph with $n$ vertices and non-negative edge lengths and let $\epsilon > 0$. There is a $(1+\epsilon)$-approximate distance oracle for $G$ with $O(1)$ query time, $O(n\log n(\log n/\epsilon + f(\epsilon)))$ size and $O(n\log n(\log^3 n/\epsilon^2 + f(\epsilon)))$ pre-processing time, where $f(\epsilon) = 2^{O(1/\epsilon)}$.*

The oracle in Theorem 1 has a constant query time independent of $\epsilon$ and size nearly linear in the graph size. This improves the query time of the previous works [15,21] that are (nearly) linear in $1/\epsilon$ for non-constant $\epsilon$. Wulff-Nilsen gives an $O(1)$ time exact distance oracle for $G$ with size $O(n^2(\log\log n)^4/\log n)$ [23]. There exists some constant $c_0 > 0$ such that for $\frac{1}{\epsilon} < c_0 \log n$, our oracle has a smaller size.

The result in Theorem 1 can be generalized to an oracle described in the next theorem.

**Theorem 2.** *Let $G$ be an undirected planar graph with $n$ vertices and non-negative edge lengths, $\epsilon > 0$ and $1 \leq \eta \leq 1/\epsilon$. There is a $(1+\epsilon)$-approximate distance oracle for $G$ with $O(\eta)$ query time, $O(n\log n(\log n/\epsilon + f(\eta\epsilon)))$ size and $O(n\log n(\log^3 n/\epsilon^2 + f(\eta\epsilon)))$ pre-processing time, where $f(\eta\epsilon) = 2^{O(1/(\eta\epsilon))}$.*

Our results build on some techniques used in the previous approximate distance oracles for planar graphs. Thorup [21] gives a $(1+\epsilon)$-approximate distance oracle for planar graph $G$ with $O(1/\epsilon)$ query time. Informally, some techniques used in the oracle are as follows: Decompose $G$ into a balanced recursive subdivision; $G$ is decomposed into subgraphs of balanced sizes by shortest paths and each subgraph is decomposed recursively until every subgraph is reduced to a pre-defined size. A path $Q$ *intersects* a path $Q'$ if $V(Q) \cap V(Q') \neq \emptyset$. A set $\mathcal{Q}$ of paths is a path-separator for vertices $u$ and $v$ if every path between $u$ and $v$ intersects a path $Q \in \mathcal{Q}$. Vertices $u$ and $v$ are *shortest-separated* by a path $Q$ if there exist a shortest path between $u$ and $v$ that intersects $Q$. If vertices $u$ and $v$ have a path-separator $\mathcal{Q}$ then $u$ and $v$ is shortest-separated by some path $Q \in \mathcal{Q}$. For each subgraph $X$ of $G$, let $\mathcal{P}(X)$ be the set of shortest paths used to decompose $X$. For each path $Q \in \mathcal{P}(X)$ and each vertex $u$ in $X$, a set $P_Q(u)$ of $O(1/\epsilon)$ vertices called *portals* on $Q$ is selected. For vertices $u$ and $v$ shortest-separated by some path $Q$ in $\mathcal{P}(X)$, $\min_{p\in P_Q(u), q\in P_Q(v), Q\in\mathcal{P}(X)} d_G(u,p) + d_G(p,q) + d_G(q,v)$ is used to approximate $d_G(u,v)$. The oracle keeps the distances $d_G(u,p)$ and $d_G(p,v)$.

The portal set $P_Q(u)$ above is vertex dependent. For a path $Q$ in $G$ of length $d(Q)$, there is a set $P_Q$ of $O(1/\epsilon)$ portals such that for any vertices $u$ and $v$ shortest-separated by $Q$, $\min_{p\in P_Q} d_G(u,p) + d_G(p,v) \leq d_G(u,v) + \epsilon d(Q)$ [17]. Based on this and a scaling technique, Kawarabayashi et al. [15] give another $(1+\epsilon)$-approximate distance oracle: Create subgraphs of $G$ such that the vertices in each subgraph satisfy certain distance property (scaling). Each subgraph $H$ of $G$ is decomposed by shortest paths into a $\rho$-division of $H$ which consists of $O(|V(H)|/\rho)$ subgraphs of $H$, each has size $O(\rho)$. For each subgraph $X$ of $H$, let $\mathcal{B}(X)$ be the set of shortest paths used to separate $X$ from the rest of $H$. For each path $Q \in \mathcal{B}(X)$, a portal set $P_Q$ is selected. For vertices $u$ and $v$ shortest-separated by some path $Q \in \mathcal{B}(X)$, $\min_{p\in P_Q, Q\in\mathcal{B}(X)} d_H(u,p) + d_H(p,v)$ is used to approximate $d_G(u,v)$. This oracle does not keep the distances $d_H(u,p)$ and $d_H(p,v)$ but uses the distance oracle in [19] to get the distances. By choosing an appropriate value $\rho$, the oracle has a better product of query time and oracle size than that of Thorup's oracle.

We also use the scaling technique to create subgraphs of $G$. We decompose each subgraph $H$ of $G$ into a balanced recursive subdivision as in Thorup's oracle. For each subgraph $X$ of $H$ and each shortest path $Q$ used to decompose $X$, we choose one set $P_Q$ of $O(1/\epsilon)$ portals on $Q$ for all vertices in $X$. A new ingredient in our oracle is to use a more time efficient data structure to approximate $d_G(u,v)$ instead of $\min_{p\in P_Q, Q\in\mathcal{P}(X)} d_H(u,p) + d_H(p,v)$. Using an approach in [22], we show that the vertices in $V(X)$ can be partitioned into $s = f(\epsilon)$ classes $A_1, ..., A_s$ such that for every two classes $A_i$ and $A_j$, there is a key portal $p_{ij} \in P_Q$ and for any $u \in A_i$ and $v \in A_j$, if $u$ and $v$ are shortest-separated by $Q$ then $d_H(u, p_{ij}) + d_H(p_{ij}, v) \le (1+\epsilon)d_G(u, v)$ and $d_H(u, p_{ij}) + d_H(p_{ij}, v)$ can be computed in $O(1)$ time. This gives a $(1+\epsilon)$-approximate distance oracle with $O(1)$ query time. Our oracle includes a data structure for some details of applying the scaling technique that are not mentioned in previous studies as well.

Our computational model is word RAM, which models what we can program using standard programming languages such as C/C++. In this model, a word is assumed big enough to store any vertex identifier or distance. We also assume basic operations, which include addition, subtraction, multiplication, bitwise operations (AND, OR, NEGATION) and left/right cyclic shift on a word have unit time cost.

The rest of the paper is organized as follows. In the next section, we give preliminaries of the paper and review the techniques on which our oracles build. In Section 3, we present distance oracles with additive stretch. In Section 4, we give the $(1+\epsilon)$-approximate distance oracles which use the additive stretch oracles as subroutines. The final section concludes the paper.

## 2. Preliminaries

An undirected graph $G$ consists of a set $V(G)$ of vertices and a set $E(G)$ of edges. For a subset $A \subseteq E(G)$, we denote by $V(A)$ the set of vertices incident to at least one edge of $A$. For $A \subseteq E(G)$ and $W \subseteq V(G)$, we denote by $G[A]$ and $G[W]$ the subgraphs of $G$ induced by $A$ and $W$, respectively. A graph $H$ is a subgraph of $G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

A path between vertices $u$ and $v$ in $G$ is a sequence of edges $e_1, .., e_k$, where $e_i = \{v_{i-1}, v_i\}$ for $1 \le i \le k$, $u = v_0$, $v = v_k$, and the vertices $v_0, ..., v_k$ are distinct. For any edge $e$, let $l(e)$ be the length of $e$. The length of path $Q = e_1, ..., e_k$ is $d(Q) = \sum_{1 \le i \le k} l(e_i)$. A path $Q$ is a shortest path between vertices $u$ and $v$ if $d(Q)$ is the minimum among those of all paths between $u$ and $v$. The distance between vertices $u$ and $v$ in $G$, denoted by $d_G(u, v)$, is the length of a shortest path between $u$ and $v$. For each vertex $u$ in $G$, the *eccentricity* of $u$ is $\lambda(u) = \max_{v \in V(G)} d_G(u, v)$. The *radius* of $G$ is $r(G) = \min_{u \in V(G)} \lambda(u)$. The *diameter* of $G$ is $d(G) = \max_{u \in V(G)} \lambda(u)$.

A graph is planar if it has a planar embedding (a drawing on a sphere without edge crossing). In the rest of this paper, graphs are undirected planar graphs with non-negative edge lengths unless otherwise stated.

A basic approach in this paper is to decompose graph $G$ into subgraphs by shortest paths. A set $\mathcal{P}$ of shortest paths in graph $G$ is a *shortest path separator* of $G$ if $G[V(G) \backslash W]$, $W = \cup_{Q \in \mathcal{P}} V(Q)$, has at least $t \ge 2$ connected nonempty subgraphs $G_1, .., G_t$ of $G$. A set $\mathcal{Q}$ of paths *separates* subgraphs $G_i$ and $G_j$, $i \ne j$, if for any vertex $u$ in $G_i$ and any vertex $v$ in $G_j$, any path in $G$ between $u$ and $v$ intersects a path in $\mathcal{Q}$. For a subgraph $G_i$ of $G$, a set $\mathcal{B}(G_i)$ of paths is a *boundary* of $G_i$ if $\mathcal{B}(G_i)$ separates $G_i$ and the rest of $G$ and for every path $Q \in \mathcal{B}(G_i)$, there is an edge connecting $Q$ and $G_i$. For $\alpha > 0$, a shortest path separator $\mathcal{P}$ of $G$ is called *$\alpha$-balanced* if $|V(G_i)| \le \alpha |V(G)|$ holds for every subgraph $G_i$. An *$\alpha$-balanced recursive subdivision* of $G$ is a structure that $G$ is decomposed into subgraphs $G_1, .., G_t$ by an $\alpha$-balanced separator and for each $1 \le i \le t$, $G_i$ is decomposed recursively until each subgraph is reduced to a pre-defined size. In the recursive decomposition of $G_i$, the subset of the shortest path separator $\mathcal{P}$ of $G$ that forms a boundary $\mathcal{B}(G_i)$ of $G_i$ is included in computing a shortest path separator of $G_i$. Let $T_r$ be a shortest path spanning tree of graph $G$ rooted at a vertex $r$. Every path in $T_r$ from the root $r$ to any vertex is a shortest path and called a *root path* in $G$. We use Thorup's method [21] to compute a $\frac{1}{2}$-balanced recursive subdivision using shortest path separators composed of root paths in $G$ (based on the result in [18], this can be done in linear time).

We now briefly describe Thorup's method. Readers may refer to Section 2.5 in [21] for more details. A recursive subdivision of $G$ can be viewed as a rooted tree $T_G$ with each vertex of $T_G$ (called a *node*, to be distinguished from a vertex of $G$) representing a subgraph of $G$ and the root node representing $G$. Each node in $T_G$ with node degree one is called a *leaf node*, otherwise an *internal node*. We identify subgraphs with their corresponding nodes in $T_G$ when convenient. For any node $X$ of $T_G$, the depth of $X$ is the number of edges of $T_G$ from $X$ to the root node. The depth of $T_G$ is the largest depth of any node in $T_G$. For each node $X$ of $T_G$, let $\mathcal{B}(X)$ be the minimum set of root paths in $G$ that forms a boundary of $X$ ($\mathcal{B}(G) = \emptyset$). Let $X \cup \mathcal{B}(X)$ denote the subgraph of $G$ induced by $V(X) \cup V(\mathcal{B}(X))$. Let $X + \mathcal{B}(X)$ denote the graph obtained by removing some vertices from $X \cup \mathcal{B}(X)$ as follows: for every vertex $v$ of $\mathcal{B}(X)$ that has degree two in $X \cup \mathcal{B}(X)$, its incident edges $(u, v)$ and $(v, w)$ are replaced by edge $(u, w)$ whose length is the sum of the length of $(u, v)$ and that of $(v, w)$. For each internal node $X$, a $\frac{1}{2}$-balanced shortest path separator $\mathcal{P}(X)$ of root paths is used to decompose $X$ into subgraphs $X_1, .., X_t$, $t \ge 2$, as follows: Let $W = V(\mathcal{P}(X))$ and $X_1^*, .., X_t^*$ be the connected components of $G[V(X + \mathcal{B}(X)) \setminus W]$. Then $E(X_i) = E(X) \cap E(X_i^*)$, $1 \le i \le t$. Note that $\mathcal{P}(X)$ separates $X_i$ from $X_j$ in $X$ and $\mathcal{B}(X) \cup \mathcal{P}(X)$ separates $X_i$ from $X_j$ in $G$ for $1 \le i, j \le t$ and $i \ne j$. We now state some important properties of the $\frac{1}{2}$-balanced recursive subdivision in the next Lemma.

**Lemma 1.** *[21] Given a graph $G$ and a shortest path spanning tree $T_r$ of $G$, a $\frac{1}{2}$-balanced recursive subdivision $T_G$ of $G$ can be computed in $O(n \log n)$ time such that for each internal node $X$ of $T_G$, $|V(X_i)| \le |V(X)|/2$ $(1 \le i \le t)$ and $|\mathcal{P}(X)| = O(1)$, and for each node $X$, $|\mathcal{B}(X)| = O(1)$. Moreover, for each node $X$ of $T_G$ and each root path $Q$ of $T_r$, if $Q \in \mathcal{B}(X)$, then $Q \in \mathcal{P}(X')$ for some ancestor $X'$ of $X$ in $T_G$.*

The recursive subdivision of $G$ in Lemma 1 will be used in our oracles. Note that since the size of a subgraph is reduced by at least a factor of 1/2, the depth of $T_G$ is bounded above by $\log n$. For every vertex $v \in V(G)$, we define the *home* of $v$, denoted by $X_v$, to be the node of $T_G$ of largest depth that contains $v$. For any $u, v \in V(G)$, we define $X_{u,v}$ to be the *nearest common ancestor* of $X_u$ and $X_v$ in $T_G$. Harel and Tarjan show in [14] that after a linear time pre-processing, the nearest common ancestor of any two nodes in a tree can be found in $O(1)$ time.

Let $Q$ be a shortest path in $G$ and $\epsilon > 0$. Thorup shows that for every vertex $u$ in $G$, there is subset $P_Q(u) \subseteq V(Q)$ of $O(1/\epsilon)$ vertices such that for any vertices $u$ and $v$ shortest-separated by $Q$

$$d_G(u,v) \leq \min_{p \in P_Q(u), q \in P_Q(v)} d_G(v,p) + d_G(p,q) + d_G(q,v) \leq (1+\epsilon) d_G(u,v).$$

The vertices of $P_Q(u)$ are called *portals* on $Q$ for $u$. For every subgraph $X$ in a $\frac{1}{2}$-balanced recursive subdivision of $G$ and every shortest path $Q \in \mathcal{B}(X) \cup \mathcal{P}(X)$, by keeping the distance from each vertex $u$ in $X$ to every portal in $P_Q(u)$ explicitly, Thorup shows the following result.

**Lemma 2.** *[21] For graph $G$ and $\epsilon > 0$, there is a $(1+\epsilon)$-approximate distance oracle with $(1/\epsilon)$ query time, $O(n \log n/\epsilon)$ size and $O(n \log^3 n/\epsilon^2)$ pre-processing time. Especially for $\epsilon = 1$, there is a 2-approximate distance oracle for $G$ with $O(1)$ query time, $O(n \log n)$ size and $O(n \log^3 n)$ pre-processing time.*

Our oracles will use this oracle for $\epsilon = 1$ (any constant works) to get a rough estimation of $d_G(u,v)$.

To reduce the query time to a constant independent of $\epsilon$, we will use a portal set $P_Q$ independent of vertex $u$. For vertices $u$ and $v$ shortest-separated by a path $Q$, $d_G(u,v) = \min_{p \in V(Q)} d_G(u,p) + d_G(p,v)$. For a $P_Q \subseteq V(Q)$, $\min_{p \in P_Q} d_G(u,p) + d_G(p,v)$ approximates $d_G(u,v)$. The following result will be used.

**Lemma 3.** *[17] For a path $Q$ in $G$, $\epsilon > 0$ and $D \geq d(Q)$, a set $P_Q$ of $O(1/\epsilon)$ vertices in $V(Q)$ can be selected in $O(|V(Q)|)$ time such that for any pair of vertices $u$ and $v$ shortest-separated by $Q$, $d_G(u,v) \leq \min_{p \in P_Q} d_G(u,p) + d_G(p,v) \leq d_G(u,v) + \epsilon D$.*

The set $P_Q$ in Lemma 3 is called the *$\epsilon$-portal set (with respect to $D$)* and every vertex in $P_Q$ is called a *portal*. Given a path $Q$ starting from a vertex $r$, $\epsilon > 0$ and $D \geq d(Q)$, $P_Q$ can be computed as follows: add $r$ to $P_Q$, traverse along $Q$ from $r$ and add a vertex $v \in V(Q)$ to $P_Q$ if $d_G(u,v) \geq \epsilon D/2$, where $u$ is the last added portal in $P_Q$. To apply the $\epsilon$-portal set to our oracle, we further need to guarantee $d_G(u,v) = \Omega(D)$ for vertices $u$ and $v$ in question. We will use the *sparse neighborhood covers* introduced in [1,2,5] of $G$ to achieve this goal.

**Lemma 4.** *[5] For $G$ and $\gamma \geq 1$, connected subgraphs $G(\gamma,1), \dots, G(\gamma,n_\gamma)$ of $G$ with the following properties can be computed in $O(n \log n)$ time:*

1. *For each vertex $u$ in $G$, there is at least one $G(\gamma,i)$ that contains $u$ and every $v$ with $d_G(u,v) \leq \gamma$.*
2. *Each vertex $u$ in $G$ is contained in at most 18 subgraphs.*
3. *Each subgraph $G(\gamma,i)$ has radius $r(G(\gamma,i)) \leq 24\gamma - 8$.*

## 3. Oracle with additive stretch

We first give a distance oracle which for any vertices $u$ and $v$ in $G$, and any $\epsilon_0 > 0$, returns $\tilde{d}(u,v)$ with $d_G(u,v) \leq \tilde{d}(u,v) \leq d_G(u,v) + 7\epsilon_0 d(G)$. Based on the scaling technique in [15] and Lemma 4, this oracle will be extended to an oracle stated in Theorem 1 for $G$ in the next section.

We start with a basic data structure which keeps the following information:

- A $\frac{1}{2}$-balanced recursive subdivision $T_G$ of $G$ as in Lemma 1, each leaf node in $T_G$ has size $O(2^{(1/\epsilon_0)})$.
- A table storing $X_v$ for every $v \in V(G)$.
- A data structure with $O(1)$ query time to find the nearest common ancestor $X_{u,v}$ of $X_u$ and $X_v$ in $T_G$ for any $u$ and $v$ in $G$.
- For each internal node $X$ of $T_G$, an $\epsilon_0$-portal set $P_Q$ for every shortest path $Q \in \mathcal{P}(X) \cup \mathcal{B}(X)$. For every $P_Q$, every $u \in V(X)$ and every portal $p \in P_Q$, distance $\hat{d}(u,p)$ with

$$d_G(u,p) \leq \hat{d}(u,p) \leq d_G(u,p) + \epsilon_0 d(G).$$

- For every leaf node $X$ and every pair of $u$ and $v$ in $X$, we keep

$$\tilde{d}(u,v) = \min\{d_X(u,v), \min_{p \in P_Q, Q \in \mathcal{B}(X)} \hat{d}(u,p) + \hat{d}(p,v)\}.$$

The data structure above gives a distance oracle with $3\epsilon_0 d(G)$ additive stretch and $O(1/\epsilon_0)$ query time: Given vertices $u$ and $v$, if $X_{u,v}$ is a leaf node then $\tilde{d}(u,v)$ can be found in $O(1)$ time. Otherwise, $u$ and $v$ must be shortest-separated by some path in $\mathcal{B}(X_{u,v}) \cup \mathcal{P}(X_{u,v})$. We define

$$\tilde{d}(u,v) = \min_{p \in P_Q, Q \in \mathcal{B}(X_{u,v}) \cup \mathcal{P}(X_{u,v})} \hat{d}(u,p) + \hat{d}(p,v).$$

Let $P = \cup_{Q \in \mathcal{B}(X_{u,v}) \cup \mathcal{P}(X_{u,v})} P_Q$ and we define

$$q = \arg_{p \in P} \min\{d_G(u,p) + d_G(p,v)\}.$$

From $\hat{d}(u,q) \le d_G(u,q) + \epsilon_0 d(G)$, $\hat{d}(q,v) \le d_G(q,v) + \epsilon_0 d(G)$ and Lemma 3,

$$\begin{aligned} d_G(u,v) &\le \tilde{d}(u,v) \le \hat{d}(u,q) + \hat{d}(q,v) \\ &\le d_G(u,q) + d_G(q,v) + 2\epsilon_0 d(G) \le d_G(u,v) + 3\epsilon_0 d(G). \end{aligned}$$

$\tilde{d}(u,v)$ can be computed in $O(1/\epsilon_0)$ time because $|P_Q| = O(1/\epsilon_0)$ and $|\mathcal{B}(X_{u,v}) \cup \mathcal{P}(X_{u,v})| = O(1)$.

We first reduce the query time for internal nodes in the above oracle to a constant independent of $\epsilon_0$ and then analyze the pre-processing time of the distance oracle. For $z > 0$, let $f(z) = 2^{O(1/z)}$. Based on an approach in [22], we show that for each internal node $X$ and each path $Q \in \mathcal{B}(X) \cup \mathcal{P}(X)$, the vertices in $V(X)$ can be partitioned into $f(\epsilon_0)$ classes such that for any two classes $A_i$ and $A_j$, there is a key portal $p_{ij} \in P_Q$ and for every $u \in A_i$ and every $v \in A_j$ shortest-separated by $Q$, $\hat{d}(u,p_{ij}) + \hat{d}(p_{ij},v) \le d_G(u,v) + 7\epsilon_0 d(G)$. By keeping the classes and key portals, the query time is reduced to $O(1)$. We first define the classes.

**Definition 1.** Let $Q$ be a shortest path in $G$, $r(G) \le D \le d(G)$ and $P_Q = \{p_1 ..., p_l\}$ be an $\epsilon_0$-portal set (with respect to $D$) on $Q$. The vertices of $G$ are partitioned into classes based on $\hat{d}(u,p_i)$, $p_i \in P_Q$ as follows. For each vertex $u$, a vector $\vec{\Gamma}_u = (a_1, ..., a_l)$ is defined such that for $1 \le i \le l$, $a_i = \left\lceil \hat{d}(u,p_i)/(\epsilon_0 D) \right\rceil$. Vertices $u$ and $v$ are in the same class if and only if $\vec{\Gamma}_u = \vec{\Gamma}_v$.

The following property of the classes defined above is straightforward.

**Property 1.** *Let $Q$ be a shortest path in $G$, $r(G) \le D \le d(G)$ and $P_Q$ be an $\epsilon_0$-portal set with respect to $D$ on $Q$. Let $A$ be any class of vertices in $G$ defined in Definition 1. For any two vertices $u, v \in A$ and any portal $p \in P_Q$, $\hat{d}(u,p) - \epsilon_0 D \le \hat{d}(v,p) \le \hat{d}(u,p) + \epsilon_0 D$.*

We show more properties of the classes defined above in the next two lemmas.

**Lemma 5.** *Let $Q$ be a shortest path in $G$, $r(G) \le D \le d(G)$ and $P_Q$ be an $\epsilon_0$-portal set with respect to $D$ on $Q$. Let $A_i$ and $A_j$ be any two classes of vertices in $G$ defined in Definition 1. There is a key portal $p_{ij} \in P_Q$ such that for any vertices $u \in A_i$ and $v \in A_j$ shortest-separated by $Q$, $d_G(u,v) \le \hat{d}(u,p_{ij}) + \hat{d}(p_{ij},v) \le d_G(u,v) + 7\epsilon_0 d(G)$.*

**Proof.** We choose arbitrarily a vertex $x \in A_i$ and a vertex $y \in A_j$. Let $p_{ij} = \arg_{p_i \in P_Q} \min\{\hat{d}(x,p_i) + \hat{d}(p_i,y)\}$ be the key portal. For any $u \in A_i$ and $v \in A_j$ shortest-separated by $Q$, let $q = \arg_{p_i \in P_Q} \min\{d_G(u,p_i) + d_G(p_i,v)\}$ and let $p = \arg_{p_i \in P_Q} \min\{\hat{d}(u,p_i) + \hat{d}(p_i,v)\}$. Then

$$\begin{aligned} \hat{d}(u,p) + \hat{d}(p,v) &\le \hat{d}(u,q) + \hat{d}(q,v) \\ &\le d_G(u,q) + d_G(q,v) + 2\epsilon_0 d(G) \le d_G(u,v) + 3\epsilon_0 d(G), \end{aligned}$$

because $\hat{d}(u,q) \le d_G(u,q) + \epsilon_0 d(G)$, $\hat{d}(q,v) \le d_G(q,v) + \epsilon_0 d(G)$, $P_Q$ is an $\epsilon_0$-portal set and Lemma 3. From $u, x \in A_i$, Property 1 and $D \le d(G)$,

$$\hat{d}(u,p_i) \le \hat{d}(x,p_i) + \epsilon_0 D \le \hat{d}(x,p_i) + \epsilon_0 d(G) \le \hat{d}(u,p_i) + 2\epsilon_0 d(G)$$

for every $p_i \in P_Q$. The same relations hold for $v, y$ because they are in $A_j$. So

$$\begin{aligned} \hat{d}(u,p_{ij}) + \hat{d}(p_{ij},v) &\le \hat{d}(x,p_{ij}) + \hat{d}(p_{ij},y) + 2\epsilon_0 d(G) \\ &\le \hat{d}(x,p) + \hat{d}(p,y) + 2\epsilon_0 d(G) \le \hat{d}(u,p) + \hat{d}(p,v) + 4\epsilon_0 d(G). \end{aligned}$$

Therefore,

$$\begin{aligned} d_G(u,v) &\le \hat{d}(u,p_{ij}) + \hat{d}(p_{ij},v) \le \hat{d}(u,p) + \hat{d}(p,v) + 4\epsilon_0 d(G) \\ &\le d_G(u,v) + 7\epsilon_0 d(G). \end{aligned}$$

This completes the proof of the lemma. □

**Lemma 6.** *The total number of classes by Definition 1 is $f(\epsilon_0)$.*

**Proof.** Essentially, this result is proved by Weimann and Yuster in [22] but somehow hidden in other details. Below we give a self-contained proof of the lemma. For each vector $\vec{\Gamma}_u = (a_1, .., a_l)$, let $\vec{\Gamma}^*_u = (a_1, (a_2 - a_1), (a_3 - a_2), .., (a_l - a_{l-1}))$. Then $\vec{\Gamma}_u = \vec{\Gamma}_v$ if and only if $\vec{\Gamma}^*_u = \vec{\Gamma}^*_v$. So we just need to prove that the total number of different $\vec{\Gamma}^*_u$ is $f(\epsilon_0)$. From Definition 1,

$$\begin{aligned}|a_i - a_{i-1}| &= \left|\left\lceil \frac{\hat{d}(u,p_i)}{\epsilon_0 D}\right\rceil - \left\lceil \frac{\hat{d}(u,p_{i-1})}{\epsilon_0 D}\right\rceil\right| \\ &\le \left|\frac{\hat{d}(u,p_i) - \hat{d}(u,p_{i-1})}{\epsilon_0 D}\right| + 1 \\ &\le \left|\frac{d_G(u,p_i) - d_G(u,p_{i-1})}{\epsilon_0 D}\right| + 2 \le \frac{d_G(p_{i-1},p_i)}{\epsilon_0 D} + 2.\end{aligned}$$

Since $P_Q$ is an $\epsilon_0$-portal set, $l = O(1/\epsilon_0)$. So

$$\sum_{2\le i\le l} |a_i - a_{i-1}| \le \frac{d_G(p_1,p_l)}{\epsilon_0 D} + 2l = O(1/\epsilon_0).$$

Therefore there are $2^{O(1/\epsilon_0)}$ different vectors of $(a_1, |a_2 - a_1|, |a_3 - a_2|, .., |a_l - a_{l-1}|)$. The $i$'th element of $(a_1, (a_2 - a_1), (a_3 - a_2), .., (a_l - a_{l-1}))$ is either $|a_i - a_{i-1}|$ or $-|a_i - a_{i-1}|$. Therefore, there are $2^{O(1/\epsilon_0)}$ different $\vec{\Gamma}^*_u$. □

Notice that we can assume that for each internal node $X$, the number of classes Definition 1 is at most $|V(X)|^2$ because otherwise, instead of partitioning the vertices into classes, we can simply use a $|V(X)| \times |V(X)|$ distance array to keep the shortest distance between every pair of vertices in $X$.

Now we are ready to show a data structure $\mathrm{DS}_0$ for our oracle with $7\epsilon_0 d(G)$ additive stretch. $\mathrm{DS}_0$ contains the basic data structure given above and the following additional information:

- For each internal node $X$ of $T_G$ and each shortest path $Q \in \mathcal{B}(X) \cup \mathcal{P}(X)$, let $A^Q_1, ..., A^Q_s$ be the classes of vertices in $V(X)$ defined in Definition 1. For each vertex $u \in V(X)$, we give an index $I^Q_X(u)$ with $I^Q_X(u) = i$ if $u \in A^Q_i$; and an $s \times s$ array $C_Q$ with $C_Q[i, j]$ containing the key portal $p^Q_{ij}$ for classes $A^Q_i$ and $A^Q_j$.

We now describe how to compute the distances $\hat{d}(d, p)$ for internal nodes as defined in the basic data structure. The method is essentially the same as in the fast construction in [21], but simpler as the portal sets we use are not vertex dependent. We use Lemma 1 to get the recursive subdivision $T_G$ of $G$. Let $T_r$ be the shortest path spanning tree of $G$ as defined in Section 2 and let $D$ be the largest length of any root path of $T_r$. By a depth first search of $T_r$ from $r$, we compute an $\epsilon_0$-portal set $P_Q$ (with respect to $D$) and an *auxiliary* $(\epsilon_0/\log n)$*-portal set* $\Gamma_Q$ (with respect to $D$) for every $Q \in \mathcal{P}(X) \cup \mathcal{B}(X)$, $X \in V(T_G)$. We compute the distances $\hat{d}(u, p)$ for every internal node $X$ in a top–down traversal on $T_G$ from root $G$. We use Dijkstra's algorithm to compute $\hat{d}(u, p)$ for every $u$ in $X$ and every $p \in P_Q \cup \Gamma_Q$, $Q \in \mathcal{P}(X)$. Recall that $X \cup \mathcal{B}(X)$ denotes the subgraph of $G$ induced by $V(X) \cup V(\mathcal{B}(X))$. Let $X \star \mathcal{B}(X)$ denote the graph obtained from $X \cup \mathcal{B}(X)$ as follows: for every $u$ in $X$ and every $p'$ in $\cup_{Q' \in \mathcal{B}(X)} P_{Q'} \cup \Gamma_{Q'}$, add edge $\{u, p'\}$ with length $\hat{d}(u, p')$; then remove every degree two vertex of $\mathcal{B}(X)$ as what we do for $X + \mathcal{B}(X)$. For the root node, the computation is on $G$. For an internal node $X \ne G$, the computation is on $X \star \mathcal{B}(X)$. Note that $X \star \mathcal{B}(X)$ may not be planar and since $|\mathcal{B}(X)| = O(1)$, $|V(X \star \mathcal{B}(X))|$ is linear in the number of edges of $G$ incident to vertices of $X$ plus the number of portals in each path. Notice that $Q'$ is in $\mathcal{P}(X')$ for some internal node $X'$ which is an ancestor of $X$ in $T_G$. So the distances $\hat{d}(u, p')$ have been computed when we construct $X \star \mathcal{B}(X)$. Note that for some vertex $u$ and portal $p$, $\hat{d}(u, p)$ may be computed multiple times. But the value of $\hat{d}(u, p)$ does not change: let $H_i, 1 \le i$, be the graph on which $\hat{d}(u, p)$ is computed for the $i$th time; $\hat{d}(u, p)$ does not increase because the edge $\{u, p\}$ is contained in $H_i$ for $i \ge 2$; and $\hat{d}(u, p)$ does not decrease because $H_{i+1}$ is a subgraph of $H_i$ for $i \ge 2$. For $X$ with $|V(X)| \ge \log n/\epsilon_0$, we run Dijkstra's algorithm using every $p \in P_Q \cup \Gamma_Q$ as the source. For $X$ with $|V(X)| < \log n/\epsilon_0$, we run Dijkstra's algorithm using every $u$ in $X$ as the source. After the distances $\hat{d}(u, p)$, $p \in P_Q \cup \Gamma_Q$, for all internal nodes have been computed, we only keep the distances $\hat{d}(u, p)$ to the portals $p \in P_Q$ for every internal node.

In the next lemma, we show that the distances $\hat{d}(u, p)$ computed above meet the requirement of $\mathrm{DS}_0$.

**Lemma 7.** *For every internal node $X$ of $T_G$, every vertex $u$ in $X$ and every portal $p \in P_Q \cup \Gamma_Q$, $Q \in \mathcal{P}(X)$, $\hat{d}(u, p) \le d_G(u, p) + \epsilon_0 d(G)$.*

**Proof.** For every internal node $X$ of depth $k$ in $T_G$, every vertex $u$ in $X$ and every portal $p \in P_Q \cup \Gamma_Q$, $Q \in \mathcal{P}(X)$, we prove by induction that $\hat{d}(u, p) \le d_G(u, p) + \frac{k\epsilon_0}{\log n} d(G)$. For the root node (of depth 0), $\hat{d}(u, p) = d_G(u, p)$ because the distances are computed on $G$. Assume that for every internal node of depth at most $k-1 \geq 0$, $\hat{d}(u,p) \leq d_G(u,p) + \frac{(k-1)\epsilon_0}{\log n} d(G)$. Let $X$ be a node of depth $k$. For $u$ in $X$ and $p \in P_Q \cup \Gamma_Q$, $Q \in \mathcal{P}(X)$, let $P(u,p)$ be a shortest path between $u$ and $p$. If $P(u,p)$ contains only edges in $X$ then $\hat{d}(u,p) = d_G(u,p)$ and the statement is proved. Otherwise, $P(u,p)$ can be partitioned into two subpaths $P(u,y)$ and $P(y,p)$, where every vertex of $P(u,y)$ except $y$ is in $X$ and $y$ is a vertex of a path $Q' \in \mathcal{B}(X)$. Note that $y$ is incident to some vertex of $X$ so $y$ appears in $X \star \mathcal{B}(X)$. From the way $\Gamma_{Q'}$ is computed and the fact that $D \leq d(G)$, where $D$ is used for computing $\Gamma_{Q'}$, there is a portal $p_y \in \Gamma_{Q'}$ such that $d_{Q'}(y,p_y) = d_G(y,p_y) \leq \frac{\epsilon_0}{2\log n} d(G)$. Let $X'$ be an ancestor of $X$ such that $Q' \in \mathcal{P}(X')$. Note that $\hat{d}(u,p_y)$ is computed in $X' \star \mathcal{B}(X')$ and that $y$, $p_y$ and $X$ (and therefore $P(u,y)$) are all contained in $X' \star \mathcal{B}(X')$. Therefore,

$$\hat{d}(u,p_y) \leq d(P(u,y)) + d_{Q'}(y,p_y) \leq d(P(u,y)) + \frac{\epsilon_0}{2\log n} d(G)$$

and

$$d_G(p_y,p) \leq d(P(y,p)) + d_G(y,p_y) \leq d(P(y,p)) + \frac{\epsilon_0}{2\log n} d(G).$$

The distance $\hat{d}(p_y,p)$ has been computed in a node $X'$ which is an ancestor of $X$ and has depth at most $k-1$. So $\hat{d}(p_y,p) \leq d_G(p_y,p) + \frac{(k-1)\epsilon_0}{\log n} d(G)$. Because edges $\{u,p_y\}$ and $\{p_y,p\}$ with lengths $\hat{d}(u,p_y)$ and $\hat{d}(p_y,p)$ are contained in the graph $X \star \mathcal{B}(X)$,

$$\begin{aligned} \hat{d}(u,p) &\leq \hat{d}(u,p_y) + \hat{d}(p_y,p) \\ &\leq d(P(u,y)) + \frac{\epsilon_0}{2\log n} + d(P(y,p)) + \frac{\epsilon_0}{2\log n} + \frac{(k-1)\epsilon_0}{\log n} d(G) \\ &= d_G(u,p) + \frac{k\epsilon_0}{\log n} d(G). \end{aligned}$$

Since each node in $T_G$ has depth at most $\log n$, $\hat{d}(u,p) \leq \epsilon_0 d(G)$. □

The next three lemmas give the pre-processing time, space requirement and query time for data structure $\mathrm{DS}_0$.

**Lemma 8.** *For graph $G$ and $\epsilon_0 > 0$, data structure $\mathrm{DS}_0$ can be computed in $O(n(\log^3 n/\epsilon_0^2 + f(\epsilon_0)))$ time.*

**Proof.** Let $T_G$ be the recursive subdivision of $G$ and $b = 2^{(1/\epsilon_0)}$. It takes $O(n\log n)$ time to compute $T_G$ (Lemma 1). It takes $O(n)$ time to compute the data structure that can answer the least common ancestor of any two nodes in $T_G$ in $O(1)$ time [14], $O(n\log n)$ time to compute $X_v$ for every $v \in V(G)$, and $O(n)$ time to compute $P_Q \cup \Gamma_Q$ for every path $Q \in \mathcal{B}(X) \cup \mathcal{P}(X)$, $X \in V(T_G)$.

For every node $X$, let $M(X)$ be the number of edges in $G$ incident to vertices in $X$. Then $\sum_{X \in T_G} M(X) = O(n\log n)$. For every path $Q$, $|P_Q \cup \Gamma_Q| = O(\log n/\epsilon_0)$ and for every node $X$ in $T_G$, $|\mathcal{B}(X) \cup \mathcal{P}(X)| = O(1)$. From this, for each internal node $X$, $X \star \mathcal{B}(X)$ has $O(M(X) + \log n/\epsilon_0)$ vertices and $O(M(X) + \log n/\epsilon_0)$ edges. Dijkstra's algorithm is executed $\min\{M(X), \log n/\epsilon_0\}$ times for each node $X$. It takes $O(M(X)(\log n/\epsilon_0)^2)$ time to compute all $\hat{d}(u,p)$ for node $X$. Since the sum of $M(X)$ for all nodes $X$ of the same depth is $O(n)$, it takes $O(n(\log n/\epsilon_0)^2)$ time for all internal nodes of the same depth. Since $T_G$ has depth $O(\log n)$, it takes $O(n\log^3 n/\epsilon_0^2)$ time to compute all distances $\hat{d}(u,p)$ for all internal nodes.

To find $\tilde{d}(u,v)$ for a leaf node $X$, we first use Dijkstra's algorithm to compute $d_X(u,v)$, taking every vertex of $X$ as the source. This takes $O(b^2\log b) = O(b^2/\epsilon_0)$ time for one leaf node since $|V(X)| = O(b)$ and $O(nb/\epsilon_0)$ time for all leaf nodes since the sum of $|V(X)|$ for all leaf nodes $X$ is $O(n)$. Then we compute

$$\tilde{d}(u,v) = \min\{d_X(u,v), \min_{p \in P_Q, Q \in \mathcal{B}(X)} \hat{d}(u,p) + \hat{d}(p,v)\}.$$

From $|P_Q| = O(1/\epsilon_0)$ for $Q \in \mathcal{B}(X)$ and $|\mathcal{B}(X)| = O(1)$, this takes $O(b^2/\epsilon_0)$ time for one leaf node and $O(nb/\epsilon_0)$ time for all leaf nodes. The total time to compute $\tilde{d}(u,v)$ for all leaf nodes is $O(nb/\epsilon_0)) = O(nf(\epsilon_0))$.

The value $D$ for computing the classes can be found in $O(n)$ time. Since there are $O(n)$ internal nodes, by Lemma 6, it takes $O(nf(\epsilon_0)(1/\epsilon_0)) = O(nf(\epsilon_0))$ time to compute all classes and key portals. Therefore, $\mathrm{DS}_0$ can be computed in $O(n(\log^3 n/\epsilon_0^2 + f(\epsilon_0)))$ time. □

**Lemma 9.** *For graph $G$ and $\epsilon_0 > 0$, the space requirement for data structure $\mathrm{DS}_0$ is $O(n(\log n/\epsilon_0 + f(\epsilon_0)))$.*

**Proof.** Let $T_G$ be the recursive subdivision of $G$ in $\mathrm{DS}_0$ and $b = 2^{(1/\epsilon_0)}$. Each leaf node $X$ has $O(b)$ vertices and requires $O(b^2)$ space to keep the distances $\tilde{d}(u,v)$ for $u, v$ in the node. From this and the sum of $|V(X)|$ for all leaf nodes is $O(n)$, the space for all leaf nodes is $O(nb) = O(nf(\epsilon_0))$. By Lemma 1, the sum of $|V(X)|$ for all nodes $X$ in $T_G$ is $O(n\log n)$. From $|\mathcal{B}(X)\cup\mathcal{P}(X)| = O(1)$ for every $X$ and $|P_Q| = O(1/\epsilon_0)$ for each $Q \in \mathcal{B}(X)\cup\mathcal{P}(X)$, the total space for keeping the distances $\hat{d}(u,v)$ between vertices and portals is $O(n\log n/\epsilon_0)$. By Lemma 6, the space for the classes $A_1^Q, .., A_s^Q$ in each internal node $X$ is $f(\epsilon_0)$ for every $Q \in \mathcal{B}(X)\cup\mathcal{P}(X)$. Since there are $O(n)$ internal nodes, the total space for the classes in all nodes is $O(nf(\epsilon_0)) = O(nf(\epsilon_0))$.

Therefore the space requirement for the oracle is $O(n(\log n/\epsilon_0 + f(\epsilon_0)))$. □

**Lemma 10.** *For graph $G$ and $\epsilon_0 > 0$, $\tilde{d}(u,v)$ with $d_G(u,v) \le \tilde{d}(u,v) \le d_G(u,v) + 7\epsilon_0 d(G)$ can be computed in $O(1)$ time for any $u$ and $v$ in $G$ using data structure* DS$_0$.

**Proof.** Let $T_G$ be the recursive subdivision of $G$ in DS$_0$. $X_u$, $X_v$ and $X_{u,v}$ can be found in $O(1)$ time. If $X_{u,v}$ is a leaf node then $\tilde{d}(u,v)$ can be found in $O(1)$ time. Otherwise, for each path $Q \in \mathcal{B}(X_{u,v})\cup\mathcal{P}(X_{u,v})$, assume that $u \in A_i^Q$ and $v \in A_j^Q$, and let $p_{ij}^Q$ be the key portal for $A_i^Q$ and $A_j^Q$. By Lemma 5,

$$\tilde{d}(u,v) = \min_{p_{ij}^Q,\, Q\in\mathcal{B}(X_{u,v})\cup\mathcal{P}(X_{u,v})} \hat{d}(u,p_{ij}^Q) + \hat{d}(p_{ij}^Q, v) \le d_G(u,v) + 7\epsilon_0 d(G).$$

Since $|\mathcal{B}(X)\cup\mathcal{P}(X)| = O(1)$ and the key portal $p_{ij}^Q$ can be found in $O(1)$ time for each path $Q \in \mathcal{B}(X)\cup\mathcal{P}(X)$, $\tilde{d}(u,v)$ can be computed in $O(1)$ time. □

From Lemmas 8, 9 and 10, we have the following result.

**Theorem 3.** *For graph $G$ and $\epsilon_0 > 0$, there is an oracle which gives a distance $\tilde{d}(u,v)$ with $d_G(u,v) \le \tilde{d}(u,v) \le d_G(u,v) + 7\epsilon_0 d(G)$ for any vertices $u$ and $v$ in $G$ with $O(1)$ query time, $O(n(\log n/\epsilon_0 + f(\epsilon_0)))$ size and $O(n(\log^3 n/\epsilon_0^2 + f(\epsilon_0)))$ pre-processing time.*

We can make the oracle in Theorem 3 a more generalized one: For integer $\eta$ satisfying $1 \le \eta \le 1/\epsilon_0$, we partition each path $Q \in \mathcal{B}(X)\cup\mathcal{P}(X)$ into $\eta$ segments $Q_1, .., Q_\eta$, compute the classes $A_1^{Q_l}, .., A_s^{Q_l}$ of vertices in $V(X)$ for each segment $Q_l$, $1 \le l \le \eta$, and key portal $p_{ij}^{Q_l}$, and use

$$\tilde{d}(u,v) = \min_{p_{ij}^{Q^l},\, 1\le l\le\eta,\, Q\in\mathcal{B}(X)\cup\mathcal{P}(X)} \hat{d}(u,p_{ij}^{Q_l}) + \hat{d}(p_{ij}^{Q_l}, v)$$

to approximate $d_G(u,v)$. By this generalization, we get the following result.

**Theorem 4.** *For graph $G$, $\epsilon_0 > 0$ and $1 \le \eta \le 1/\epsilon_0$, there is an oracle which gives a distance $\tilde{d}(u,v)$ with $d_G(u,v) \le \tilde{d}(u,v) \le d_G(u,v) + 7\epsilon_0 d(G)$ for any vertices $u$ and $v$ in $G$ with $O(\eta)$ query time, $O(n(\log n/\epsilon_0 + f(\eta\epsilon_0)))$ size and $O(n(\log^3 n/\epsilon_0^2 + f(\eta\epsilon_0)))$ pre-processing time.*

## 4. Oracle with $(1+\epsilon)$ stretch

For $\epsilon > 0$, by choosing an $\epsilon_0 = \frac{\epsilon}{7c}$ where $c > 0$ is a constant, the oracle in Theorem 3 gives a $(1+\epsilon)$-approximate distance oracle for graph $G$ with $d_G(u,v) \ge d(G)/c$ for every $u$ and $v$ in $G$. For graph $G$ with $d_G(u,v)$ much smaller than $d(G)$ for some $u$ and $v$, we use a scaling approach as described in [15] to get a $(1+\epsilon)$-approximate distance oracle. The idea is to compute a set of oracles as described in Theorem 3, each for a computed subgraph $H$ of $G$. Given $u$ and $v$, we can find in $O(1)$ time a constant number of subgraphs (and the corresponding oracles) such that the minimum value returned by these oracles is a $(1+\epsilon)$-approximation of $d_G(u,v)$. Therefore a $(1+\epsilon)$-approximate distance for any $u, v$ can be computed in constant time. We assume $\epsilon > 5/n$, otherwise a naive exact distance oracle with $O(1)$ query time and $O(n^2)$ space can be used to prove Theorem 1.

Let $l_m$ be the smallest edge length in $G$. We assume $l_m \ge 1$ and the case where $l_m < 1$ can be easily solved in a similar way by normalizing the length of each edge $e$ of $G$ to $l(e)/l_m$. Following the scaling technique in [15], for each scale $\gamma \in \{2^i | 0 \le i \le \lceil \log d(G) \rceil\}$, we contract every edge $e$ of length $l(e) < \gamma/n^2$ in $G$ and remove edges $e$ with $l(e) > 24\gamma$, self loops and degree 0 vertices to get a contracted graph $G_\gamma$, and then compute a sparse cover $\mathcal{C}_\gamma = \{G(\gamma, j), j = 1, ..., n_\gamma\}$ of $G_\gamma$ as in Lemma 4. When an edge $e = \{u,v\}$ is contracted in a graph, $\{u,v\}$ is removed, vertices $u$ and $v$ are replaced by a new vertex $w$, and every edge other than $\{u,v\}$ incident to $u$ or $v$ in the graph is made incident to $w$. We say the new vertex $w$ covers vertices $u$ and $v$. We say a vertex $u$ covers $u$ itself and if a vertex $x$ covers a vertex $w$ then $x$ covers every vertex covered by $w$. We say an edge $\{u,v\}$ of $G$ appears in a scale $\gamma$ if $G_\gamma$ has an edge $\{x,y\}$ such that $x$ and $y$ cover $u$ and $v$, respectively. We say a vertex $x$ appears in scale $\gamma$ if $G_\gamma$ contains $x$. From the construction of $G_\gamma$, each edge of $G$ only appears in scales $\gamma$ satisfying $l(e)/24 \le \gamma \le l(e)n^2$ and thus appears in $O(\log n)$ different scales. A scale $\gamma$ is *non-trivial* if $G_\gamma$ has at least one edge. Since each edge of $G$ appears in $O(\log n)$ scales, there are $O(n\log n)$

non-trivial scales and the total number of vertices appearing in each non-trivial scale is $O(n\log n)$. We can have a bijection $\phi: \{\gamma|\gamma$ is a non-trivial scale.$\} \rightarrow \{0, 1, .., N\}$, where $N = O(n\log n)$, such that for non-trivial scales $\gamma' < \gamma$, $\phi(\gamma') < \phi(\gamma)$. By the bijection $\phi$, we can assume that non-trivial scales are $0, 1, .., N$. In what follows, we use *scale* for a non-trivial scale.

Our $(1+\epsilon)$-approximate distance oracle first estimates roughly the distance $d(u, v)$ to get a right scale $\gamma$ and then uses oracles $\mathrm{DS}_0(\gamma, j)$ and the vertices in subgraphs $G(\gamma, j)$ that cover $u$ and $v$ to find an approximate distance $\tilde{d}(u, v)$. It is not trivial to find the vertices in $G(\gamma, j)$ that cover $u$ and $v$ in $O(1)$ time and $\tilde{O}(n)$ space. A simple approach for finding the vertices in $G(\gamma, j)$ that cover $u$ and $v$ in $O(1)$ time is to keep a pointer from each vertex $u$ of $G$ to every vertex covering $u$, but this requires $O(n^2\log n)$ space, too large. To reduce the space, instead of keeping a pointer from $u$ to every vertex covering $u$, we create a rooted tree $T_C$ of size $O(n\log n)$ on the vertices in each scale such that if vertex $x$ covers vertex $u$ then $x$ is an ancestor of $u$ in $T_C$. We further create a data structure of size $O(|T_C|)$ (called Find-Ancestor) which, given a scale $\gamma$ and a vertex $u$ of $G$ that $u$ is covered by the vertex $x$ appearing in scale $\gamma$, answers $x$ in $O(1)$ time by finding the ancestor $x$ of $u$ in $T_C$. A vertex $w$ (either a vertex of $G$ or one from the contraction of edges) may appear in multiple scales. We denote by $w(\gamma)$ vertex $w$ appearing in scale $\gamma$. For $\gamma \neq \gamma'$, $w(\gamma)$ and $w(\gamma')$ are distinct vertices in $T_C$. The construction of $T_C$ is as follows:

1. Initially, each vertex $u$ of $G$ is given a scale label $sl(u) = 0$.
2. For each scale $\gamma = 1, 2, .., N$ and each vertex $w(\gamma)$, if $w(\gamma)$ is a new vertex by the contraction of edges $e_1, .., e_k$ then for each vertex $u \in \cup_{1\le i\le k} e_i$, include edge $\{w(\gamma), u(\gamma')\}$ in $T_C$, where $\gamma' = sl(u)$, otherwise include edge $\{w(\gamma), w(\gamma')\}$, $\gamma' = sl(w)$, in $T_C$; update $sl(w)$ to $\gamma$.
3. After Step 2, $T_C$ is a forest. Create a vertex $r$ and an edge between $r$ and the root of each tree in the forest to get a tree $T_C$ with root $r$.

Since each vertex of $T_C$ except the root $r$ is a vertex of $G_\gamma$ and each edge of $G$ appears in $O(\log n)$ scales, the size of $T_C$ is $O(n\log n)$. There is a bijection between the set of leaf vertices of $T_C$ and $V(G)$. For each vertex $x$ of $T_C$, each ancestor of $x$ covers $x$ in some $G_\gamma$. Bender and Farach-Colton [4] give a data structure which, given any vertex $x$ in a rooted tree of $n$ vertices and an integer $d$ no larger than the depth of $x$ in the tree, find in $O(1)$ time the ancestor of $x$ with depth $d$. Given a vertex $u$ of $G$ and a scale $\gamma$, the data structure in [4] provides a base to find the ancestor $w(\gamma)$ of $u$ in $T_C$.

Our problem of finding the ancestor $w(\gamma)$ of $u$ is slightly different from the one in [4] as we do not know the depth of $w(\gamma)$ and we only need to find $w(\gamma)$ for a leaf of $T_C$. As in [4], we decompose $T_C$ into disjoint paths: find a longest root-leaf path $P$ in $T_C$ and remove $P$ from $T_C$; the removal of $P$ breaks the remaining of $T_C$ into subtrees $T_1, T_2, ..$; these subtrees are decomposed recursively by removing the longest root-leaf paths in the subtrees to get a set of disjoint paths. Each leaf $u$ of $T_C$ is in exactly one path and each path has exactly one leaf. We denote by $P_u$ the path having leaf $u$. For each vertex $x$ in $T_C$, let $d(x)$ denote the depth of $x$ (the number of edges in the path between $x$ and $r$ in $T_C$). For a leaf $u$ of $T_C$ and $0 \le k \le d(u)$, an ancestor $x$ of $u$ is the $k$th ancestor of $u$ if the length of the path between $u$ and $x$ is $k$. For a path $P_u$ of length $h$, we create a ladder $Q_u$ such that $Q_u$ is the path in $T_C$ between $u$ and its $k$th ancestor, where $k = \min\{2h-1, d(u)-1\}$. Notice that each vertex of $T_C$ is in one root-leaf path but may be in multiple ladders. For each vertex $v$ in $P_u$, we say that the ladder $Q_u$ created for $P_u$ is $v$'s ladder. For each leaf $u$ of $T_C$, we create edges (short cuts) $e_i, i = 0, 1, ...$ such that $e_i$ connects $u$ and the $2^i$th ancestor $x_i$ of $u$. We will use the following result of [4].

**Fact 1.** *[4] For every k with $2^i \le k < 2^{i+1}$, the kth ancestor x of u is in $x_i'$s ladder.*

For each vertex $u$ of $T_C$, the depths of all ancestors of $u$ are consecutive integers $0, 1, .., d(u)$. By this property, as shown in [4], it is easy to find $u'$s ancestor $x_i$ from the given depth $d$ in $O(1)$ time such that $x_i'$s ladder has $u'$s ancestor $x$ of depth $d$; further, each ladder can be simply put into an array and $x$ can be found in $O(1)$ time from the array for $x_i'$s ladder using $d$ as an index of the array.

Given a vertex $u$ of $G$ and a scale $\gamma$, we need to find the ancestor $w(\gamma)$ of $u$ in $T_C$ using $\gamma$ instead of the depth of $w(\gamma)$. As in [4], we also first find a correct $x_i$ and then $w(\gamma)$ from $x_i'$s ladder. But our tasks are more complex than those in [4] because the scales of all ancestors of $u$ may not be consecutive integers and we can not find the depth of $w(\gamma)$ in $O(1)$ time from $\gamma$. To find $x_i$, for each leaf $u$ of $T_C$, let $S_u$ be the set of scales of $u'$s ancestors $x_i$ connected to $u$ by the short cuts of $u$. Then $S_u$ is a subset of $\{0, 1, .., N\}$. Since the length of a root-leaf path in $T_C$ is at most $N$, there are $O(\log N)$ short cuts for each leaf of $T_C$ and $|S_u| = O(\log N)$. For $|S_u| = O(\log N)$, Fredman and Willard [11] give a data structure of size $O(|S_u|)$ that given a scale $\gamma \in \{0, 1, .., N\}$ finds in $O(1)$ time the largest scale $\gamma'$ with $\gamma' \le \gamma$ (the predecessor of $\gamma$) in $S_u$. We create the data structure of [11] for every $S_u$. Vertex $w(\gamma)$ is the $k$th $(k = d(u) - d(w(\gamma)))$ ancestor of $u$. If $\gamma = 0$ then $w(\gamma) = u$. Otherwise, for the largest $\gamma' \in S_u$ with $\gamma' \le \gamma$, the ancestor $w(\gamma')$ of $u$ is $x_i$ such that $2^i \le d(u) - d(w(\gamma)) < 2^{i+1}$. By Fact 1, $w(\gamma)$ is in $x_i'$s ladder. Since we can find $\gamma'$ in $O(1)$ time, we can find $x_i$ in $O(1)$ time. To find $w(\gamma)$ from $x_i'$s ladder, for each ladder $Q_u$, let $R_u$ be the set of scales of the vertices in $Q_u$. We create a hash table of [10] with size $O(|R_u|)$ for all scales in $R_u$ that, given a scale $\gamma$ in $R_u$, answers in $O(1)$ time the vertex $w(\gamma)$ in $Q_u$.

Our data structure Find-Ancestor consists of the data structure of ladders from $T_C$ and short-cuts for each leaf of $T_C$, the data structure for finding the predecessor of a given scale $\gamma$ from $S_u$ (thus a correct $x_i$) and the hash table for finding $w(\gamma)$ in $x_i'$s ladder. The following result is a summary of the above.

**Lemma 11.** *Given a vertex $u$ of $G$ and a scale $\gamma$, data structure Find-Ancestor finds the ancestor $w(\gamma)$ of $u$ in $O(1)$ time and $O(n \log n)$ space. The data structure can be computed in $O(n \log^4 n)$ time.*

**Proof.** Obviously, the data structure finds $w(\gamma)$ in $O(1)$ time. The rooted tree $T_C$ has size $O(n \log n)$. The size of all ladders is $O(n \log n)$ since each ladder $Q_u$ only doubles the size of the root-leaf path $P_u$. The size of short cuts for all $u$ of $G$ is $O(n \log n)$ because each $u$ has $O(\log N) = O(\log n)$, $N = O(n \log n)$, short cuts. For each $u$ of $G$, $|S_u| = O(\log n)$ and $\sum_{u \in V(G)} |S_u| = O(n \log n)$. From this, the size of the data structure for finding the predecessor in $S_u$ for all $S_u$ is $O(n \log n)$ as the size for each $|S_u|$ is $O(|S_u|)$. The size of the hash tables for all ladders is $O(n \log n)$ since the size of the hash table for each $Q_u$ is $O(|Q_u|)$ and the size of all ladders is $O(n \log n)$. $T_C$, the ladders and short cuts can be computed in $O(n \log n)$ time. The data structure for one $S_u$ can be computed in $O(|S_u|^4) = O(\log^4 N) = O(\log^4 n)$ time [11] and thus the data structure for all $S'_u$s can be computed in $O(n \log^4 n)$ time. The hash table for one ladder $Q_u$ can be computed in $O(|Q_u| \log^2 n)$ time [3] and thus the hash tables for all ladders can be computed in $O(n \log^2 n)$ time. □

The data structure $\mathrm{DS}_1$ for our $(1+\epsilon)$-approximate distance oracle keeps the following information:

- A 2-approximate distance oracle $\mathrm{DS}_T$ of $G$ in Lemma 2.
- For every scale $\gamma$, subgraphs $G(\gamma, j)$ and for each subgraph $G(\gamma, j)$, an oracle $\mathrm{DS}_0(\gamma, j)$ in Theorem 3 with $\epsilon_0 = \epsilon / c'$, $c' > 0$ is a constant to be specified below.
- For every scale $\gamma$ and every vertex $x$ appearing in scale $\gamma$, the index $j$ of every subgraph $G(\gamma, j)$ that contains $x$.
- Data structure Find-Ancestor.

**Lemma 12.** *For graph $G$ and $\epsilon > 0$, $\tilde{d}(u, v)$ with $d_G(u, v) \le \tilde{d}(u, v) \le (1+\epsilon) d_G(u, v)$ can be computed in $O(1)$ time for any $u$ and $v$ in $G$ using data structure $\mathrm{DS}_1$.*

**Proof.** Given vertices $u$ and $v$ in $G$, oracle $\mathrm{DS}_T$ gives $\tilde{d}_T(u, v)$ with $d_G(u, v) \le \tilde{d}_T(u, v) \le 2 d_G(u, v)$ in $O(1)$ time (Lemma 2). If $\tilde{d}_T(u, v) = 0$ then 0 is returned as $d_G(u, v)$. Otherwise, given $\tilde{d}_T(u, v)$, a scale $\gamma$ with $\gamma/2 < \tilde{d}_T(u, v) \le \gamma$ can be found by computing the most significant bit of $\left\lceil \tilde{d}_T(u, v) \right\rceil$. In the word RAM model with unit costs for basic operations, this can be computed in $O(1)$ time using the fusion tree data structure proposed in [11]. Notice that each of $u$ and $v$ is covered by a vertex in scale $\gamma$ and let $x$ and $y$ be the vertices in scale $\gamma$ covering $u$ and $v$, respectively. By Lemma 11, $x$ and $y$ can be computed in $O(1)$ time. By Lemma 4, there is a $G(\gamma, j)$ that contains $x$ and every $w$ with $d_{G_\gamma}(x, w) \le \gamma$ and $d(G(\gamma, j)) = O(\gamma) = O(d_G(u, v))$. Therefore, there exists a constant $c_1 > 0$ (96 would do) such that $d(G(\gamma, j)) \le c_1 d_G(u, v)$. It is easy to see that $\mathrm{DS}_0(\gamma, j)$ returns a minimum distance among all the oracles at this scale containing $x, y$. By oracle $\mathrm{DS}_0(\gamma, j)$, we get a distance $\tilde{d}_0(x, y)$ with $d_{G(\gamma,j)}(x, y) \le \tilde{d}_0(x, y) \le d_{G(\gamma,j)}(x, y) + 7\epsilon_0 d(G(\gamma, j))$. Since $G(\gamma, j)$ is a subgraph of $G_\gamma$ which is obtained by contracting every edge $e$ of $G$ with $l(e) < \gamma/n^2$, $d_{G(\gamma,j)}(x, y) \le d_G(u, v)$. Let $L$ be the largest sum of the lengths of the contracted edges in any path in $G$. Then $d_G(u, v) \le d_{G(\gamma,j)}(x, y) + L$ and $L < \gamma/n$. From $\gamma < 2\tilde{d}_T(u, v) \le 4 d_G(u, v)$ and $\epsilon > 5/n$, $L < \gamma/n \le \frac{4}{5}\epsilon d_G(u, v)$. Let $\tilde{d}(u, v) = \tilde{d}_0(x, y) + \gamma/n$. Then

$$\begin{aligned} d_G(u, v) \le \tilde{d}(u, v) &= \tilde{d}_0(x, y) + \gamma/n \le d_{G(\gamma,j)}(x, y) + 7\epsilon_0 d(G(\gamma, j)) + \gamma/n \\ &\le d_G(u, v) + 7c_1 \frac{\epsilon}{c'} d_G(u, v) + \frac{4}{5}\epsilon d_G(u, v). \end{aligned}$$

By choosing $c' = 35 c_1$, we have $d_G(u, v) \le \tilde{d}_G(u, v) \le (1+\epsilon) d_G(u, v)$. By Lemma 2, it takes $O(1)$ time to compute $\tilde{d}_T(u, v)$. Given $\gamma$, $u$ and $v$, the vertex $x$ covering $u$ and vertex $y$ covering $v$ can be found in $O(1)$ time. From Lemma 4, there are $O(1)$ graphs $G(\gamma, j)$ containing $x$ and $y$. From this and Theorem 3, it takes $O(1)$ time to compute $\tilde{d}(u, v)$. □

**Lemma 13.** *Data structure $\mathrm{DS}_1$ requires $O(n \log n (\log n/\epsilon + f(\epsilon)))$ space and can be computed in $O(n \log n (\log^3 n/\epsilon^2 + f(\epsilon)))$ time.*

**Proof.** $\mathrm{DS}_T$ requires $O(n \log n)$ space. Each $\mathrm{DS}_0(\gamma, j)$ requires $O(n_{\gamma j} \log n_{\gamma j}/\epsilon + n_{\gamma j} f(\epsilon))$ space, where $n_{\gamma j} = |V(G(\gamma, j))|$. Each edge $e$ of $G$ appears in $O(\log n)$ different scales and in each scale $\gamma$, $e$ appears in $O(1)$ subgraphs $G(\gamma, j)$. From this, $\sum_{\gamma, j} n_{\gamma j} = O(n \log n)$. For every scale $\gamma$, $\mathrm{DS}_1$ keeps every vertex $x$ in scale $\gamma$ and index $j$ of every subgraph $G(\gamma, j)$ that contains $x$. This requires $O(n \log n)$ space since each edge of $G$ appears in $O(\log n)$ scales and each $x$ appears in $O(1)$ subgraphs. By Lemma 11, data structure Find-Ancestor has size $O(n \log n)$. Therefore, $\mathrm{DS}_1$ requires space $O(n \log n (\log n/\epsilon + f(\epsilon)))$.

$DS_T$ can be computed in $O(n \log^3 n)$ time and the sparse neighborhood covers can be computed in $O(n \log^2 n)$ time. The time for computing $\mathrm{DS}_0(\gamma, j)$ for each $G(\gamma, j)$ is $O(n_{\gamma j} \log^3 n_{\gamma j}/\epsilon^2 + f(\epsilon))$. By Lemma 11, data structure Find-Ancestor can be computed in $O(n \log^4 n)$ time. Therefore, $\mathrm{DS}_1$ can be computed in $O(n \log n (\log^3 n/\epsilon^2 + f(\epsilon))$ time. □

From Lemmas 12 and 13, we get Theorem 1 which is restated below.

**Theorem 1.** *For $\epsilon > 0$, there is a $(1+\epsilon)$-approximate distance oracle for $G$ with $O(1)$ query time, $O(n \log n(\log n/\epsilon + f(\epsilon)))$ size and $O(n \log n(\log^3 n/\epsilon^2 + f(\epsilon)))$ pre-processing time.*

Using the oracle in Theorem 4 instead of $DS_0$, we get Theorem 2.

## 5. Concluding remarks

It is open whether there is a $(1+\epsilon)$-approximate distance oracle with $O(1)$ query time and size nearly linear in $n$ for weighted directed planar graphs. For undirected planar graphs, it is interesting to reduce oracle size and pre-processing time (the function $f(\epsilon)$) for the oracles in this paper. Experimental studies for fast query time distance oracles are worth investigating.

## Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments. Especially, the data structure Find-Ancestor is a result for answering the question of a reviewer on the details of applying the scaling technique. The research was partially supported by the Canada NSERC Discovery Grant No. 250304.

## References

[1] B. Awerbuch, B. Berger, L. Cowen, D. Peleg, Near-linear time construction of sparse neighborhood covers, SIAM J. Comput. 28 (1) (1998) 263–277.
[2] B. Awerbuch, D. Peleg, Sparse partitions, in: Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science, FOCS 1990, 1990, pp. 503–513.
[3] N. Alon, M. Naor, Derandomization, witnesses for Boolean matrix multiplication, and construction of perfect hash functions, Algorithmica 16 (1996) 434–449.
[4] M.A. Bender, M. Farach-Colton, The level ancestor problem simplified, Theoret. Comput. Sci. 321 (2004) 5–12.
[5] C. Busch, R. LaFortune, S. Tirthapura, Improved sparse covers for graphs excluding a fixed minor, in: Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, 2007, pp. 61–70.
[6] S. Chechik, Approximate distance oracle with constant query time, in: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC 2014, 2014, pp. 654–663.
[7] V. Cohen-Added, S. Dahlgaad, C. Wulff-Nilsen, Fast and compact distance oracle for planar graphs, in: Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, 2017, pp. 962–973.
[8] N. Delling, A. Goldberg, T. Pajor, R. Werneck, Robust Exact Distance Queries on Massive Networks, Microsoft Research Technique Report, MSR-TR-2014-12, 2014.
[9] H. Djidjev, Efficient algorithms for shortest path problems on planar graphs, in: Proceedings of the 22nd International Workshop on Graph-Theoretical Concepts in Computer Science, WG 1996, 1996, pp. 151–165.
[10] M.L. Fredman, J. Komlos, E. Szemeredi, Storing a sparse table with $O(1)$ worst case access time, J. ACM 31 (3) (1984) 538–544.
[11] M.L. Fredman, D.E. Willard, Surpassing the information theoretic bound with fusion trees, J. Comput. System Sci. 47 (3) (1993) 424–436.
[12] P. Gawrychowski, S. Mozes, O. Weimann, C. Wulff-Nilsen, Better tradeoffs for exact distance oracles in planar graphs, in: Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, 2018, pp. 515–529.
[13] Q. Gu, G. Xu, Constant query time $(1+\epsilon)$-approximate distance oracle for planar graphs, in: Proceedings of the 26th International Symposium on Algorithms and Computation, ISAAC 2015, in: LNCS, vol. 9472, 2015, pp. 625–636.
[14] D. Harel, R. Tarjan, Fast algorithms for finding nearest common ancestor, SIAM J. Comput. 13 (1984) 338–355.
[15] K. Kawarabayashi, C. Sommer, M. Thorup, More compact oracles for approximate distances in undirected planar graphs, in: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, 2013, pp. 550–563.
[16] P.N. Klein, Preprocessing an undirected planar network to enable fast approximate distance queries, in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, 2002, pp. 820–827.
[17] P.N. Klein, S. Subramanian, A fully dynamic approximation scheme for shortest paths in planar graphs, Algorithmica 22 (3) (1998) 235–249.
[18] R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36 (2) (1979) 177–189.
[19] S. Mozes, C. Sommer, Exact distance oracles for planar graphs, in: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, 2012, pp. 209–222.
[20] C. Sommer, Shortest-path queries in static networks, ACM Comput. Surv. 46 (4) (2014) 45.
[21] M. Thorup, Compact oracles for reachability and approximate distances in planar digraphs, J. ACM (JACM) 51 (6) (2004) 993–1024.
[22] O. Weimann, R. Yuster, Approximating the diameter of planar graphs in near linear time, in: Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, ICALP 2013, in: LNCS, vol. 7965, 2013, pp. 828–839.
[23] C. Wulff-Nilsen, Algorithms for Planar Graphs and Graphs in Metric Spaces, Ph.D. Thesis, University of Copenhagen, 2010.
[24] C. Wulff-Nilsen, Approximate distance oracles for planar graphs with improved query time–space tradeoff, in: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, 2016, pp. 351–362.