# Faster Approximate Diameter and Distance Oracles in Planar Graphs

**Timothy M. Chan**[1] **and Dimitrios Skrepetos**[2]

1 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
tmc@illinois.edu

2 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
dskrepet@uwaterloo.ca

## Abstract

We present an algorithm that computes a $(1+\varepsilon)$-approximation of the diameter of a weighted, undirected planar graph of $n$ vertices with non-negative edge lengths in $O\left(n \log n \left(\log n + (1/\varepsilon)^5\right)\right)$ expected time, improving upon the $O\left(n\left((1/\varepsilon)^4 \log^4 n + 2^{O(1/\varepsilon)}\right)\right)$-time algorithm of Weimann and Yuster [ICALP 2013]. Our algorithm makes two improvements over that result: first and foremost, it replaces the exponential dependency on $1/\varepsilon$ with a polynomial one, by adapting and specializing Cabello's recent abstract-Voronoi-diagram-based technique [SODA 2017] for approximation purposes; second, it shaves off two logarithmic factors by choosing a better sequence of error parameters during recursion.

Moreover, using similar techniques, we improve the $(1+\varepsilon)$-approximate distance oracle of Gu and Xu [ISAAC 2015] by first replacing the exponential dependency on $1/\varepsilon$ on the preprocessing time and space with a polynomial one and second removing a logarithmic factor from the preprocessing time.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** planar graphs, diameter, abstract Voronoi diagrams

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2017.25

## 1 Introduction

In this paper we study the problem of computing the diameter of a weighted, undirected planar graph of $n$ vertices with non-negative edge lengths[1], defined as the longest shortest path distance between two vertices of the graph. Since Frederickson in 1983 [7] solved the problem in $O\left(n^2\right)$ time (by determining the all-pairs shortest paths distance matrix and returning the largest value therein), a natural question arose as to whether the diameter can be computed in subquadratic time. Poly-logarithmic speedups were given by Chan [5] in 2006 and by Wulff-Nilsen [22] in 2010; the algorithm of the former works for the unweighted case and requires $O\left(n^2 \log\log n/\log n\right)$ time; the algorithm of the latter requires the same amount of time for the unweighted case and $O\left(n^2(\log\log n)^4/\log n\right)$ time for the weighted. However, a truly subquadratic algorithm, i.e., an algorithm running in $O\left(n^{2-\delta}\right)$ time for some constant $\delta > 0$, still eluded researchers for many years.

Thus, not surprisingly, the dearth of truly subquadratic algorithms led to the consideration of approximation algorithms. A $c$-approximation of the diameter, $\Delta$, of a graph is

[1] For the rest of the introduction, we assume, unless otherwise stated, that all the discussed graphs are weighted, undirected planar graphs with non-negative edge lengths.

a value $\widetilde{\Delta}$ such that $\Delta \le \tilde{\Delta} \le c\Delta$. Using the linear-time SSSP algorithm of Henzinger et al. [12] one can trivially compute a 2-approximation. The first non-trivial approximation result was given by Berman et al. [2] in 2007; their algorithm requires $O\left(n^{3/2}\right)$ time and gives a 3/2-approximation. Weimann and Yuster [21] in 2012, in a breakthrough, presented an algorithm computing a $(1+\varepsilon)$-approximation of the diameter in near-linear time, namely $O\left(n\left((1/\varepsilon)^4 \log^4 n + 2^{O(1/\varepsilon)}\right)\right)$. Nevertheless, their solution did not settle the problem completely because the running time has *exponential* dependency on $1/\varepsilon$. Another problem with their solution is the multiple (four) logarithmic factors.

Unexpectedly, the next result came in the context of exact algorithms. In 2017, Cabello [3] (full paper in [4]) made headway, by giving the first exact truly subquadratic algorithm, requiring $\tilde{O}\left(n^{11/6}\right)$ expected time. The techniques used by Cabello are as interesting as the result itself, as he used a seemingly alien concept to planar graphs, *abstract Voronoi diagrams*, originating from computational geometry.

Cabello's seminal result bifurcates the study of the diameter problem into two main avenues. First, one could try to improve its running time. This has been partially treated in a recent paper by Gawrychowski et al. [8], who presented an algorithm requiring $\tilde{O}\left(n^{5/3}\right)$ worst-case time. No lower bound is available presently, but Cabello [4] conjectured that the diameter cannot be computed exactly in time faster than $O\left(n^{1+\delta}\right)$, for some constant $\delta > 0$. Second, one could try to use some of the techniques in Cabello's paper to approximate the diameter.

In this paper we take the second avenue. Namely, we improve the running time of Weimann and Yuster [21] by eliminating the $2^{O(1/\varepsilon)}$ factor. To do this, we adapt Cabello's technique involving abstract Voronoi diagrams. It turns out that a much simplified version of his technique is sufficient for approximation purposes, and can be combined nicely with Weimann and Yuster's algorithm. Our contribution however does not stop here; we also eliminate two of the four $\log n$ factors along the way, by using a better sequence of error parameters in the recursion from Weimann and Yuster's algorithm. Our main result is summarized by the following theorem.

▶ **Theorem 1** (Diameter). *Given a weighted, undirected planar graph of n vertices with non-negative edge lengths, we can compute a $(1+\varepsilon)$-approximation of its diameter in expected $O\left(n \log n \left(\log n + (1/\varepsilon)^5\right)\right)$ time.*

Another important problem in planar graphs is the construction of efficient $(1+\varepsilon)$-approximate distance oracles, i.e., data structures that in a query for a pair of vertices $u, v$ of a planar graph $G$, return a value $\tilde{d}$ such that $d_G(u,v) \le \tilde{d} \le (1+\varepsilon) d_G(u,v)$, where $d_G(u,v)$ is the shortest path distance from $u$ to $v$ in $G$. Thorup [20] presented a $(1+\varepsilon)$-approximate distance oracle, requiring $O\left((1/\varepsilon)^2 n \log^3 n\right)$ preprocessing time, $O\left((1/\varepsilon) n \log n\right)$ space, and $O(1/\varepsilon)$ query time, later simplified by Klein [15]. Kawarabayashi et al. [14] improved the dependency on $1/\varepsilon$ of the space-query time product from $1/\varepsilon^2$ to $1/\varepsilon$. Gu and Xu [9] combined the ideas of those results with the techniques of the diameter algorithm of Weimann and Yuster [21] to obtain the first distance oracle with constant query time (independent of both $n$ and $\varepsilon$); it requires $O\left(n \log n \left((1/\varepsilon)^2 \log^3 n + 2^{O(1/\varepsilon)}\right)\right)$ preprocessing time and $O\left(n \log n \left((1/\varepsilon) \log n + 2^{O(1/\varepsilon)}\right)\right)$ space.

Using similar techniques as the ones for our diameter result, we can also improve the $(1+\varepsilon)$-approximate distance oracle of Gu and Xu [9]; namely, we eliminate the exponential dependency on $1/\varepsilon$ on the preprocessing time and space and at the same time remove a logarithmic factor from the preprocessing time.

▶ **Theorem 2** (Distance Oracle). *Given a weighted, undirected planar graph of n vertices with non-negative edge lengths, we can construct a $(1+\varepsilon)$-approximate distance oracle, requiring*

*$O(1)$ query time, $O\left(n \log n \left(\log n \left(\log n + (1/\varepsilon)^5\right) + (1/\varepsilon)^6\right)\right)$ expected preprocessing time, and $O\left(n \log n \left(\log n + (1/\varepsilon)^6\right)\right)$ space.*

Throughout the paper we operate under the standard RAM model of computation. Let $[W] = \{1, \ldots, W\}$. We assume that all the planar graphs under discussion have a fixed, combinatorial embedding and are triangulated.

## 2 A Streamlined Version of Cabello's Technique

The main purpose of this section is to construct the following farthest neighbor data structure, which will be crucial in obtaining our diameter result in Section 3:

▶ **Theorem 3** (Farthest neighbor). *Let $H$ be a weighted, undirected planar graph of $n$ vertices with non-negative edge lengths and $W$ be an integer. Let $X$ be a set of $b$ vertices on the boundary of the outer face of $H$. Let $H^+$ be the graph obtained by adding to $H$ a vertex $z_0$, with an edge from $z_0$ to each vertex $x \in X$ of unspecified length.*

*We can preprocess $H$ in $O\left(nb^3W^2\right)$ expected time, such that the following query can be answered in $O(b \log b)$ time: given lengths drawn from $[W]$ for the $b$ edges $z_0x$ $(x \in X)$, find the distance to the farthest neighbor of $z_0$ in $H^+$, i.e., compute $\max_{u \in V(H)} d_{H^+}(z_0, u)$.*

In our application, $b = W = O(1/\varepsilon)$, so the preprocessing time would be near linear in $n$. Cabello established a similar theorem [4, Theorem 21] for the more general setting where each edge $z_0x$ $(x \in X)$ may have a real length, but his preprocessing time bound is $\tilde{O}\left(n^2b^3 + b^4\right)$. We show that when the length of each edge $z_0x$ $(x \in X)$ is a small integer, the preprocessing time can be greatly improved, and at the same time the method becomes simpler.

To avoid degeneracies we need to ensure uniqueness of the shortest paths. That can be done by perturbing the lengths of the edges of $H$ with known techniques (e.g., see [11]). Note that we do not need to perturb the weights of the sites, which remain integers.

### 2.1 Defining Voronoi diagrams in planar graphs

The general concept of abstract Voronoi diagrams in $\mathbb{R}^2$ was defined by Klein [17]. Cabello [4] applied the concept to planar graphs with weighted sites. We reiterate here the main definitions for the sake of completeness.

Each *site* $s$ of a Voronoi diagram in a planar graph is a pair $(v_s, w_s)$, where $v_s$ is the site's placement, i.e., a vertex of the graph, and $w_s$ is its weight. Given a graph $G$ and a set of sites $S$, the *Voronoi region* of a site $s \in S$ is defined as $\mathrm{VR}_G(s, S) = \{u \in V(G) \mid d_G(v_s, u) + w_s \leq d_G(v_t, u) + w_t, \forall t \in S - \{s\}\}$, i.e., as the set of all vertices closer to $s$ than to any other site under the weighted metric; the *Voronoi diagram* of $S$ is defined as $\mathrm{VD}_G(S) = \mathbb{R}^2 \setminus \bigcup_{s \in S} \mathrm{VR}_G(s, S)$.

A key concept in Voronoi diagrams is bisectors. The bisector of two sites $s$ and $t$, $\mathrm{bis}_G(s, t)$, is defined as the set of the duals of the edges in $E_G(s, t) = \{uv \in E(G) \mid d_G(u, v_s) + w_s \leq d_G(u, v_t) + w_t$ and $d_G(v, v_t) + w_t \leq d_G(v, v_s) + w_s\}$, i.e., the bisector contains the duals of all the edges whose endpoints are not both closer to the same site. Let $\{p, q\}$ be a generic (i.e., for each $u \in V(G)$ we have $d_G(u, v_p) + w_p \neq d_G(u, v_q) + w_q$) and independent (i.e., each Voronoi region is non-empty) set of sites on the boundary of the *outer* face of a planar graph $G$. Then the bisector of $p$ and $q$ is a simple cycle in the dual, passing through the dual vertex, $v_\infty$, of the outer face ([4, Lemma 5]).

As Cabello [4] showed, a Voronoi diagram in a planar graph for $b$ sites on the boundary of the outer face fulfills Klein's axioms of abstract Voronoi diagrams [17], so it can be represented abstractly as a collection of *Voronoi vertices* and *Voronoi edges*, forming a planar graph itself of size $O(b)$. A Voronoi edge corresponds to a simple path in the dual (subpath of a bisector), and a Voronoi vertex corresponds to the meeting point of three Voronoi edges.

## 2.2 Computing abstract Voronoi diagrams in planar graphs

Since abstract Voronoi diagrams can be constructed efficiently by an existing algorithm by Klein et al. [18] based on randomized incremental construction, we have:

▶ **Theorem 4** (Abstract Voronoi diagram construction). *We can construct the abstract Voronoi diagram in a planar graph with $b$ sites on the outer face, using an expected $O(b \log b)$ number of elementary operations. Here, an elementary operation refers to the computation of the abstract Voronoi diagram of any four sites.*

To prove Theorem 3, we need to construct the abstract Voronoi diagram in the graph $H$ for the $b$ sites at $X$ quickly for any given assignment of weights on $X$ from $[W]$, after an initial preprocessing that does not depend on the weights. By Theorem 4, it suffices to show how to compute the abstract Voronoi diagram of any four such sites.

We start by showing how to compute all different bisectors, given two vertices of $X$ as placements of sites, whose weights are drawn from $[W]$, by building upon [4, Lemma 17]. We need $O(nW)$ total time for constructing the bisectors, whereas Cabello needed $O\left(n^2\right)$ time for general real weights; we can return a pointer to a bisector in $O(1)$ time instead of $O(\log n)$ time.

▶ **Lemma 5** (Bisectors). *Given two vertices $v_s, v_t \subseteq X$ as placements of sites, the family of bisectors $\mathrm{bis}_H((v_s, w_s), (v_t, w_t))$, over all possible weights $w_s$ and $w_t$ drawn from $[W]$, has at most $O(W)$ different bisectors. We can compute all these bisectors in $O(nW)$ total time, such that, given two weights $w_s, w_t \in [W]$, we can return a pointer to the relevant bisector in $O(1)$ time.*

**Proof.** Assuming w.l.o.g. that $w_s \geq w_t$, we can write $\mathrm{bis}_H((v_s, w_s), (v_t, w_t))$ as $\mathrm{bis}_H((v_s, w), (v_t, 0))$, where $w = w_s - w_t$, so we need to consider only $\mathrm{bis}_H((v_s, w), (v_t, 0))$, where $w \in [W]$. Hence, there can be at most $O(W)$ different bisectors for a pair of sites.

Let $s = (v_s, w_s), t = (v_t, w_t)$, and $S = \{s, t\}$. For each vertex $u \in V(H)$ we compute the value $\eta_u = d_H(v_t, u) - d_H(v_s, u)$, by first running the linear-time SSSP algorithm of Henzinger et al. [12] from $v_s$ and $v_t$ and then visiting each vertex; $u \in V(H)$ belongs to $\mathrm{VD}_H(s, \{s, t\})$ when $w \leq \eta_u$ and to $\mathrm{VD}_H(t, \{s, t\})$ otherwise. For each $w \in [W]$ we compute the bisector $\mathrm{bis}_H((v_s, w), (v_t, 0))$, by marking each edge $uv \in E(G)$ such that $w \leq \eta_u$ and $w > \eta_v$. The bisector is composed of the duals of the marked edges and is a cycle in the dual, passing through $v_\infty$; we represent it as a linked list, $\mathrm{LL}_{s,t,w}$. Finally, we store the linked-list representation of every different bisector $\mathrm{bis}_H((v_s, w), (v_t, 0))$ in a table, $\mathrm{T}_{s,t}$, indexed by $w$.

The total time spent is $O(nW)$ because we have at most $W$ different bisectors, and computing each takes $O(n)$ time. Given two sites $s$ and $t$ with weights $w_s$ and $w_t \in [W]$ respectively, we can return a pointer to the pertinent bisector in $O(1)$ time by looking up $\mathrm{T}_{s,t}[w]$, assuming w.l.o.g. that $w_s \geq w_t$. ◀

Next, we show how to compute all different Voronoi diagrams, given three vertices of $X$ as placements of sites, whose weights are drawn from $[W]$, by building upon [4, Lemma 18]. We need $O\left(nW^2\right)$ time, whereas Cabello had $O\left(n^2\right)$; we can return a pointer to a Voronoi diagram in $O(1)$ time instead of $O(\log n)$. Furthermore, our proof is simpler since it does not involve line arrangements and amortization.

▶ **Lemma 6** (Abstract Voronoi diagrams of three sites). *Given three vertices $v_s, v_t, v_q \subseteq X$ as placements of sites, the family of Voronoi diagrams over all possible weights $w_s, w_t, w_q \in [W]$, has at most $O\left(W^2\right)$ different Voronoi diagrams. We can compute all these Voronoi diagrams in $O\left(nW^2\right)$ total time, such that given weights $w_s, w_t, w_q \in [W]$ we can return a pointer to the relevant Voronoi diagram in $O(1)$ time.*

**Proof.** We invoke Lemma 5 to compute and store all the different bisectors of each pair of the three sites in $O(nW)$ time. We can assume w.l.o.g. that $w_q = 0$, so there are at most $O\left(W^2\right)$ different Voronoi diagrams. Let $s = (v_s, w_s), t = (v_t, w_t), q = (v_q, w_q)$, and $S = \{s, t, q\}$. For each vertex $u \in V(H)$ we compute the values $\eta_x^{st} = d_H(v_s, u) - d_H(v_t, u)$, $\eta_x^{qt} = d_H(v_q, u) - d_H(v_t, u)$, and $\eta_x^{sq} = d_H(v_s, u) - d_H(v_q, u)$ by running the SSSP algorithm of [12]; $u$ belongs to $\mathrm{VR}_H(s, \{s, t, q\})$ if $\eta_u^{st} \leq w_t - w_s$ and $\eta_u^{sq} \leq -w_s$; similar statements can be made for $\mathrm{VR}_H(t, \{s, t, q\})$ and $\mathrm{VR}_H(q, \{s, t, q\})$.

For every $w_s, w_t \in [W]$, we find in linear time the Voronoi diagram $\mathrm{VD}_H(S)$ as follows. Each bisector (i) does not participate at all, (ii) participates wholly, or (iii) only a subpath of it, passing through $v_\infty$, participates in $\mathrm{VD}_H(S)$ (that is implied by that fact that $\mathrm{VD}_H(S)$ has at most one vertex besides $v_\infty$; see [4, Lemma 13]). We provide two pointers for each bisector, which mark the bisector's part that constitutes a Voronoi edge: one for its first and one for its last edge participating in $\mathrm{VD}_H(S)$. Starting from $v_\infty$, we scan the edges of each bisector in clockwise order; the first pointer of $\mathrm{bis}_H(s, t)$ is created for the first encountered edge $uv$ such that $u$ is closer to $s$ than to $t$ and to $q$, and $v$ is closer to $t$ or $q$ than to $s$ (which can be determined using their $\eta$ values). The second pointer is created for the last such edge. If no such edges are encountered, both pointers are set to NULL. We can find in $O(1)$ time if there exists a Voronoi vertex; to do so, we scan each triple of pointers of the bisectors to see if the corresponding edges meet at a common dual vertex. If that is the case, we set that vertex to be a Voronoi vertex.

The representation of $\mathrm{VD}_H(S)$ for weights $w_s, w_t \in [W]$ is composed of (i) a linked list of each bisector participating in it, (ii) the first and last pointers of each such bisector, and (iii) the one, if any, Voronoi vertex therein, besides $v_\infty$. Each different Voronoi diagram is stored in a two-dimensional table $\mathrm{T}_{s,t,q}$, indexed by $w_s$ and $w_t$. Given weights $w_s, w_t$, and $w_q$, where w.l.o.g $w_s, w_t \geq q_q$ we can return a pointer to the pertinent Voronoi diagram in $O(1)$ time by looking up $\mathrm{T}_{s,t,q}[w_s - w_q, w_t - w_q]$, assuming w.l.o.g that $w_s \geq w_q$ and $w_t \geq w_q$. ◀

The final step before using Theorem 4 is to provide a data structure that, given four vertices of $X$ as placements of sites, whose weights are drawn from $[W]$, returns their Voronoi diagram. The following lemma builds upon [4, Lemma 19]; we refer the reader therein for the proof. Our preprocessing time is $O\left(nb^3W^2\right)$, whereas Cabello had $O\left(n^2b^3\right)$; also our query time is $O(1)$ instead of $O(\log n)$.

▶ **Lemma 7** (Abstract Voronoi diagrams of four sites). *We can construct a data structure, such that (i) its preprocessing time is $O\left(nb^3W^2\right)$, and (ii) for any four vertices of $X$ as placements of sites that are generic, independent and have weights drawn from $[W]$, their abstract Voronoi diagram can be computed in $O(1)$ time.*

Now we can use Lemma 7 and Theorem 4 to compute the abstract Voronoi diagram of $b$ sites, given all the vertices of $X$ as placements of sites, whose weights are drawn from $[W]$. Our preprocessing time is $O\left(nb^3W^2\right)$ expected, whereas Cabello had $O\left(n^2b^3\right)$; our query time is $O(b \log b)$, while Cabello had multiple $\log n$ factors.

▶ **Theorem 8** (Abstract Voronoi diagrams in planar graphs). *Let $H, X, n, b,$ and $W$ be as in Theorem 3. We can preprocess $H$ in $O\left(nb^3W^2\right)$ time, such that, given the vertices of $X$ as placements of sites, whose weights are drawn from $[W]$, we can compute the Voronoi diagram of the sites in $O(b \log b)$ expected time.*

**Proof.** We first construct the data structure of Lemma 7, which after $O\left(nb^3W^2\right)$ preprocessing time can compute the abstract Voronoi diagram of any set of four sites in $O(1)$ time. Then, using Theorem 4, we compute the abstract Voronoi diagram of the $b$ sites in $O(b \log b)$ time. ◀

## 2.3 Constructing the farthest neighbor data structure

As one last ingredient for our farthest neighbor data structure, we need the following lemma, taken almost verbatim from [4, Corollary 6].

▶ **Lemma 9.** *Let $F$ be an undirected planar graph of $n$ vertices, each having a* cost $c(u) > 0$, *and let $q_0$ be one of them. Let $\Pi = \{\pi_1, \ldots, \pi_\ell\}$ be a family of simple paths in the dual of $F$ with a total of $h$ edges, counted with multiplicity. After $O(n+h)$ preprocessing time, we can answer the following query in $O(k)$ time: given a $q_0$-star-shaped cycle $\gamma$ in the dual, i.e., a cycle such that (i) $q_0$ is in the interior of $\gamma$, and (ii) for every vertex in the interior of $\gamma$ its shortest path to $q_0$ is fully contained in $\gamma$, described as a concatenation of $k$ subpaths from* $\Pi$, *return* $\max_{u \in U} \left(V_{\text{int}}\left(\gamma, F\right)\right)$*, where $V_{\text{int}}(\gamma, F)$ is the set of vertices of $F$ enclosed by $\gamma$.*

We can now prove the main theorem of this section.

**Proof of Theorem 3.** We construct the data structure of Theorem 8 for $H$, for a set $S$ of $b$ sites where each one is placed in a different vertex of $X$ and apply Lemma 5 to compute the bisector of each pair of sites. For each site $s \in S$ and each bisector $\text{bis}_H(s, \cdot)$, we assign a cost to every vertex $u \in V(H)$, equal to $d_H(v_s, u)$, and construct the data structure of Lemma 9 (a bisector $\text{bis}_H(s,t)$ enclosing $s$ is an $s$-star shaped cycle in the dual), where $F = H$, $\Pi$ is the set of bisectors, $\ell = bW$, $h = nbW$, and $k = b$. The preprocessing time is $O\left(nb^3W^2\right)$.

In a query, we compute the abstract Voronoi diagram of $H$, where for each $s \in S$ we set $w_s$ to be equal to the given length of the edge $w_0v_s$, by using the data structure of Theorem 8. For each site $s \in S$, we query the data structure of Lemma 9 for $s$ to find the vertex of $\text{VR}_H(s,S)$ with the largest distance from $s$ by walking along its boundary, which is the concatenation of at most $b$ subpaths of the bisectors $\text{bis}_H(s, \cdot)$. From Lemma 9 we need $O(b)$ time to find $\max_{u \in \text{VR}_H(s,S)}\{d_H(v_s, u) + w_s\}$. We return the maximum of those distances. Thus, the total query time is $O(b \log b)$. ◀

## 3 Improving Weimann and Yuster's Diameter Approximation Algorithm

For approximating the diameter, we employ the recursive scheme of Weimann and Yuster [21], which is as follows.

Let $\mathcal{G}$ be the original graph and $N$ its size. Let $d\left(G_1, G_2, G_3\right)$ denote the longest shortest path distance between a marked vertex of $G_1$ and a marked vertex of $G_2$ in $G_3$. Initially $G = \mathcal{G}$, $n$ is the size of $G$, all vertices are marked, and we want to approximate $d\left(G, G, G\right)$. Let $\varepsilon > 0$. The outline of the recursive scheme is as follows.

1. Find a cycle $C$ of $G$, such that the removal of $C$'s vertices decomposes $G$ into two disjoint and connected planar graphs $A$ and $B$, each having between $n/3$ and $2n/3$ vertices; $C$ may have up to $n$ vertices. Let $G_{\text{in}} = A \cup C$ and $G_{\text{out}} = B \cup C$ and assume w.l.o.g. that $A$ (resp. $B$) lies inside (resp. outside) $C$.

2. Approximate $d\left(G_{\text{in}}, G_{\text{out}}, G\right)$ (Sections 2.1 and 2.2 in [21]).
3. Unmark all vertices of $C$ and build graphs $G^+_{\text{in}}$ and $G^+_{\text{out}}$ such that (i) they are planar, and connected graphs, (ii) each of $G^+_{\text{in}}$ and $G^+_{\text{out}}$ has at most roughly $2n/3$ vertices, (iii) together they have at most roughly $n$ vertices, and (iv) $d\left(G_{\text{in}}, G_{\text{in}}, G^+_{\text{in}}\right)$ is a $(1+\varepsilon')$-approximation of $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$ for an appropriate choice of parameter $\varepsilon'$. Then recurse in $G^+_{\text{in}}$ to approximate $d\left(G_{\text{in}}, G_{\text{in}}, G^+_{\text{in}}\right)$ and do the same for $G^+_{\text{out}}$ (Section 2.3 in [21]).
4. Return $\max\left\{d\left(G_{\text{in}}, G_{\text{out}}, G\right), d\left(G_{\text{in}}, G_{\text{in}}, G^+_{\text{in}}\right), d\left(G_{\text{out}}, G_{\text{out}}, G^+_{\text{out}}\right)\right\}$.

## 3.1 Decomposing $G$ to $G_{\text{in}}$ and $G_{\text{out}}$

To decompose the graph $G$ into two subgraphs $G_{\text{in}}$ and $G_{\text{out}}$, we use a shortest path separator, similarly to Thorup's work [20]. We compute the shortest path tree $T$ of an arbitrarily selected marked vertex $z$ of $G$ in linear time by employing the algorithm of Henzinger et al. [12]. Let $\widetilde{\Delta} = \max_{u\in V(G)} d_G(v,u)$; we know that $\widetilde{\Delta} \le \Delta \le 2\widetilde{\Delta}$. We can find in linear time (see [19, Lemma 2]) two paths $P$ and $Q$, both starting at $v$, such that the removal of the vertices on $V(C)$, where $C = P \cup Q$, from $G$ gives us two disjoint planar subgraphs $A$ and $B$, where $V(A)$ (resp. $V(B)$) contains the vertices of $V(G)$ that are strictly inside (resp. outside) $C$ and $|V(A)|, |V(B)| \le 2n/3$. The size of $C$, however, can be as big as $n$. The graph $G_{\text{in}}$ (resp. $G_{\text{out}}$) is the graph induced by $A \cup C$ (resp. $B \cup C$). The time to decompose the graph is $O(n)$.

## 3.2 Reducing $d\left(G_{\text{in}}, G_{\text{out}}, G\right)$ to $d\left(G_{\text{in}}, G_{\text{out}}, G_{\text{p}}\right)$

Before approximating $d\left(G_{\text{in}}, G_{\text{out}}, G\right)$ we need to address the following issue. A shortest path between a marked vertex of $G_{\text{in}}$ and another in $G_{\text{out}}$ has to go through a vertex of $C$. However, since $C$ can have as many as $n$ vertices, we cannot consider for each such pair all the vertices of $C$; instead, we select only a small subset of vertices of $C$ and construct a graph $G_{\text{p}}$ that allows us to approximate the distance between every aforementioned pair. The following lemma can be found in [21, Section 2.1 and Lemma 2.1].

▶ **Lemma 10.** *We can select a set $Y$ of $O(1/\varepsilon)$ vertices (called* portals*) on $C$ in linear time, such that if $d\left(G_{\text{in}}, G_{\text{out}}, G\right) \ge \widetilde{\Delta}$, then $\max_{u\in V(G_{\text{in}}), v\in V(G_{\text{out}})} \min_{y\in Y}\{d_G(u,y) + d_G(y,v)\}$ is a $(1+2\varepsilon)$-approximation of $d\left(G_{\text{in}}, G_{\text{out}}, G\right)$. Otherwise, it is at most $(1+2\varepsilon)\widetilde{\Delta}$.*

We run an SSSP algorithm from each portal of $Y$ in $G$; let $\ell$ be the largest distance found. We construct a graph $G_{\text{p}} = G_{\text{p,in}} \cup G_{\text{p,out}}$, such that $V(G_{\text{p,in}}) = V(A) \cup Y$ and $V(G_{\text{p,out}}) = Y \cup V(B)$. We create an edge between each vertex of $G_{\text{p,out}}$ and each portal, whose length is equal to their shortest path distance in $G$, after rounding it to the closest multiple of $\varepsilon\ell$ and dividing it by that number. The edges between vertices of $G_{\text{p,in}}$ are the same as in $G$, but their lengths are also divided by $\varepsilon\ell$. The total time for the reduction is $O((1/\varepsilon)n)$.

## 3.3 Approximating $d\left(G_{\text{in}}, G_{\text{out}}, G_{\text{p}}\right)$

We construct the farthest neighbor data structure of Theorem 3 for $H = G_{\text{in}}$, $X = Y$, $b = O(1/\varepsilon)$, and $W = 1/\varepsilon$. Then, we query it $n$ times, by using each vertex $u \in V(G_{\text{out}})$ as $z_0$ and setting $w(z_0, x) = d_{G_{\text{out}}}(u, x)$ for each $x \in X$, to find its farthest neighbor among the vertices of $V(G_{\text{in}})$. Finally, we return the maximum of the distances found, multiplied by $\varepsilon\ell$. The total time for approximating $d\left(G_{\text{in}}, G_{\text{out}}, G_p\right)$ is thus $O\left(nb^3W^2 + nb\log b\right) = O\left((1/\varepsilon)^5 n\right)$.

Weimann and Yuster's paper [21] did not use a farthest neighbor data structure but instead employed a brute-force search, observing that there are only $2^{O(1/\varepsilon)}$ combinatorially different vertices in $A$ and $B$ (in terms of their vectors of distances to the portals). This is where we eliminate the exponential dependency on $\varepsilon$ from their algorithm.

## 3.4 Reducing $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$ to $d\left(G_{\text{in}}, G_{\text{in}}, G^{+}_{\text{in}}\right)$

After approximating $d\left(G_{\text{in}}, G_{\text{out}}, G\right)$, we need to approximate $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$ and $d\left(G_{\text{out}}, G_{\text{out}}, G\right)$; however, we cannot directly recurse in $G_{\text{in}}$ and $G_{\text{out}}$ respectively because two problems arise (since the treatment is symmetrical for both $G_{\text{in}}$ and $G_{\text{out}}$, we concern ourselves only with the former). First, a path realizing $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$ could have a subpath lying in $G_{\text{out}}$. Second, $G_{\text{in}}$ can have up to $O(n)$ vertices because $C$, which is part of $G_{\text{in}}$, itself could have that many. However, $G_{\text{in}}$ has at most $2n/3$ marked vertices, which are the only ones used for computing $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$. Therefore, we need to construct graphs $G^{+}_{\text{in}}$ and $G^{+}_{\text{out}}$ such that (i) they are planar, and connected graphs, (ii) each of $G^{+}_{\text{in}}$ and $G^{+}_{\text{out}}$ has at most roughly $2n/3$ vertices, (iii) together they have at most roughly $n$ vertices, and (iv) $d\left(G_{\text{in}}, G_{\text{in}}, G^{+}_{\text{in}}\right)$ is a $(1+\varepsilon')$-approximation of $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$ for an appropriate choice of parameter $\varepsilon'$. The following lemma is from [21, Lemma 2.3].

▶ **Lemma 11.** *If $d\left(G_{\text{in}}, G_{\text{in}}, G\right) \geq \widetilde{\Delta}$, then $d\left(G_{\text{in}}, G_{\text{in}}, G^{+}_{\text{in}}\right)$ is a $(1+2\varepsilon')$-approximation of $d\left(G_{\text{in}}, G_{\text{in}}, G\right)$. Otherwise, $d\left(G_{\text{in}}, G_{\text{in}}, G^{+}_{\text{in}}\right) \leq (1+2\varepsilon')\widetilde{\Delta}$.*

As in the algorithm of Weimann and Yuster, to construct $G^{+}_{\text{in}}$, we start by umarking all vertices of $C$ and selecting $O(1/\varepsilon')$ vertices therein, called *dense portals*, similarly to Section 3.2. Let $B_{\text{in}}$ be the union of the shortest paths between every pair of dense portals in $G_{\text{out}}$. We produce a graph $B'_{\text{in}}$, where we keep all the vertices of $B_{\text{in}}$ of degree more than two and shrink the rest. Since there are $O(1/\varepsilon')$ dense portals, there are $O\left((1/\varepsilon')^2\right)$ such shortest paths. Also, any pair of those paths shares at most one subpath since we assume that shortest paths are unique, so there are at most $O\left((1/\varepsilon')^4\right)$ vertices of $B_{\text{in}}$ of degree more than two, i.e., $V(B'_{\text{in}}) = O\left((1/\varepsilon')^4\right)$. It remains to show (i) how to compute $B'_{\text{in}}$ and (ii) how to set $\varepsilon'$. These two points are where we deviate from the approach of Weimann and Yuster.

First, to construct $B'_{\text{in}}$, we do not construct $B_{\text{in}}$ explicitly, as their algorithm does, which would require $O\left((1/\varepsilon')n\right)$ time. By using a slightly modified version of the multiple-source shortest paths data structure of Klein [16] we construct $B'_{\text{in}}$ in $O\left(n\log n + (1/\varepsilon')^4 \log n\right)$ time instead. Second, Weimann and Yuster chose a fixed value for $\varepsilon'$ for every recursive call. Since the recursion has $O(\log N)$ levels and error accumulates, they were forced to set $\varepsilon' = \varepsilon/\log N$, and so the $(1/\varepsilon')^4$ factors in the running time resulted in four $\log N$ factors. Here, we make $\varepsilon'$ adaptive, i.e., dependent on the current input size $n$. Specifically, we set $\varepsilon' = \varepsilon/n^{1/8}$. With this choice of $\varepsilon'$ we show that the approximation factor of our algorithm remains $1 + O(\varepsilon)$ (Section 3.5) and the final running time has only two $\log N$ factors (Section 3.5).

▶ **Theorem 12.** *We can build the graph $B'_{\text{in}}$ in $O\left(n\log n + (1/\varepsilon')^4 \log n\right)$ time.*

**Proof.** We apply the multiple-source shortest paths data structure of Klein, which preprocesses a planar graph $F$ of $n$ vertices in $O(n\log n)$ time, such that given a vertex $u$ on the boundary of the outer face and another vertex $v$, we can query the shortest path tree of $u$ to find the distance to $v$ in $O(\log n)$ time. We need to augment that data structure to also support the following two queries on the shortest path tree of a vertex on the boundary of the outer face: (i) find the lowest common ancestor of any two vertices; and (ii) find the level ancestor of any vertex and any level. To do that, we just replace the (persistent) dynamic trees used internally in Klein's data structure with the (persistent) *top-tree* structures of Alstrup et al. [1], so we can support both queries in $O(\log n)$ time. We construct the data structure for $F = G_{\text{out}}$, after redrawing in linear time such that $C$ lies on the outer face.

We build a list $\Gamma$ that contains all the vertices of $G_{\text{out}}$ that have degree more than three in $B_{\text{in}}$; as argued before, $|\Gamma| = O\left((1/\varepsilon')^4\right)$. There are three possibilities for each pair of shortest paths between dense portals: the paths do not intersect, they intersect only at one vertex, or they share a common subpath starting on a vertex $p_1$ and ending at another vertex $p_2$. Our goal is to find for each such pair the vertices $p_1$ and $p_2$ (which may not exist, may be the same, or may be distinct) and insert them to $\Gamma$. We focus on finding $p_1$ since finding $p_2$ is similar. Suppose that the first shortest path is from $a$ to $b$ and the second from $c$ to $d$. What makes the problem nontrivial is that the two paths are not available explicitly.

We find $p_1$ by performing a binary search on the $a$-to-$b$ shortest path as follows. Let $p'$ and $p''$ be initially set to $a$ and $b$ respectively. Let $p$ be the vertex midway between $p'$ and $p''$ on the $a$-to-$b$ path, which can be found by a level ancestor query. We want to find whether $p$ is (i) between $p_1$ and $p_2$ (i.e., on the $c$-to-$d$ path), (ii) between $a$ and $p_1$, or (iii) between $p_2$ and $b$. To do so, we find the lowest common ancestor *lca* of $p$ and $d$ on the shortest path tree of $c$. If $lca = p$, then we are in case (i). Else, we perform a level ancestor query for $p$ and $d$ to find the children $\hat{p}$ and $\hat{d}$ of *lca* that lie on the *lca*-to-$p$ and *lca*-to-$d$ paths respectively and compare the order of $\hat{p}$ and $\hat{d}$ around *lca*. We assume w.l.o.g. that $c$ is between $a$ and $b$ in $P$. If $\hat{p}$ is to the left of $\hat{d}$, then we are in case (ii), else we are in case (iii).[2] For case (i) or (iii) we recurse with $p'' = p$; for case (ii) we recurse with $p' = p$. We stop when $p' = p''$.

Once we are done with every pair of paths, we shrink every vertex in $V(G_{\text{out}}) - \Gamma$, thus procuring $B'_{\text{in}}$. ◀

We unmark all vertices of $B'_{\text{in}}$ and append it to $G_{\text{in}}$ to create the graph $G'_{\text{in}}$, which is a planar graph and has $|V(G_{\text{in}})| + O\left((1/\varepsilon')^4\right)$ vertices. Then, we have to shrink $G'_{\text{in}}$, such that it will have at most $2n/3$ vertices (remember that $|V(G_{\text{in}})|$ could be as big as $O(n)$). As in [21], we walk down on $C$ and do the following steps. For any consecutive pair $y_i$ and $y_{i+1}$ of dense portals on $C$ we create an edge between them of weight equal to their shortest path distance in $G$. Then we visit all the vertices $p_1, \ldots, p_k$ between $y_i$ and $y_{i+1}$ on $C$. For each vertex $u$ having an edge to such a vertex, we create an edge between $u$ and $y_i$ of weight equal to $\min_j\{\ell(u, p_j) + d_G(p_j, y_i)\}$. Finally, we delete all vertices $p_1, \ldots, p_k$ and their incident edges. We call the resulting graph $G^+_{\text{in}}$; it has $2n/3 + O\left(1/(\varepsilon')^4\right)$ vertices. $G^+_{\text{out}}$ is constructed similarly. The total time spent is $O\left(n \log n + (1/\varepsilon')^4 \log n\right)$.

## 3.5 Analyzing the approximation factor and the running time

▶ **Lemma 13.** *The approximation factor of our algorithm is $1 + O(\varepsilon)$.*

**Proof.** Let $G^{(\mu)}$ be the graph of a node $\mu$ of the recursion tree, and $G^{(\mu)}_{\text{in}}$ and $G^{(\mu)}_{\text{out}}$ be the two graphs created by decomposing it, as in Section 3.1. We $(1 + O(\varepsilon))$-approximate $d\left(G^{(\mu)}_{\text{in}}, G^{(\mu)}_{\text{out}}, G^{(\mu)}\right)$ for each node $\mu$ of the recursion tree, so the approximation factor of our algorithm is $(1+O(\varepsilon)) \max_\mu \frac{d\left(G^{(\mu)}_{\text{in}}, G^{(\mu)}_{\text{out}}, G^{(\mu)}\right)}{d\left(G^{(\mu)}_{\text{in}}, G^{(\mu)}_{\text{out}}, \mathcal{G}\right)} \le (1+O(\varepsilon)) \prod_i (1+\varepsilon_i)$ where $\varepsilon_i = \varepsilon/n_i^{1/8}$,

[2] If the shortest path tree is not unique, we pick the right-most one; see [16] for details.

for some sequence $n_1, n_2, \dots, n_k$ satisfying $n_{i-1}/3 + \Theta((1/\varepsilon_i)^4) \le n_i \le 2n_{i-1}/3 + \Theta((1/\varepsilon_i)^4)$ with $n_1 = N$ and $n_k = O\left((1/\varepsilon)^4\right)$.

Now, $\prod_i (1+\varepsilon_i) \le \exp\left(\sum_i \varepsilon_i\right)$. Since $n_i$ decreases at least exponentially, $\varepsilon_i$ grows at least exponentially; thus the sum $\sum_i \varepsilon_i$ is similar to a geometric series and can be bounded by the last term, which is $O(\varepsilon)$. Therefore, the approximation factor of our algorithm is $(1+O(\varepsilon))(1+O(\varepsilon)) = 1+O(\varepsilon)$ (which can be refined to $1+\varepsilon$ after adjusting $\varepsilon$ by a constant factor). ◀

▶ **Lemma 14.** *The running time of our algorithm is* $O\left(N \log N \left(\log N + (1/\varepsilon)^5\right)\right)$.

**Proof.** The running time satisfies the following recurrence relation:

$$\begin{aligned} T(n) \quad \le \quad & \max_{1/3 \le \alpha \le 2/3} \left(T\left(\alpha n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + T\left((1-\alpha)n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + \right. \\ & \left. O\left(n\left(\log n + (1/\varepsilon)^5\right)\right)\right). \end{aligned}$$

In the base case $n = O\left((1/\varepsilon)^4\right)$, so we can run a quadratic-time APSP algorithm in $O\left(n^2\right)$ time. Since we have $O\left(\varepsilon^4 N\right)$ such graphs, the total time for the base case is $O\left((1/\varepsilon)^4 N\right)$. The solution of the recurrence is $T(N) = O\left(N \log N \left(\log N + (1/\varepsilon)^5\right)\right)$. ◀

This completes the proof of Theorem 1. It is not difficult to see that in the same amount of time we can also compute a $(1+\varepsilon)$-approximation of the radius and of the Wiener index of the graph and of the eccentricity of each node.

## 4 Conclusion

Gawrychowski et al. [8] recently improved Cabello's algorithm [4] for computing the exact diameter in planar graphs; their algorithm is *deterministic* instead of randomized and requires $\tilde{O}(n^{5/3})$ time instead of $\tilde{O}(n^{11/6})$. It is worth investigating whether the techniques therein could be used to make our approximation algorithm deterministic and perhaps shave off some $1/\varepsilon$ factors. Another possible research direction is generalizing the techniques for the case of directed graphs.

An interesting consequence of our result is that we can compute the *exact* diameter of an *unweighted* planar graph in $O\left(n \log n \left(\log n + \Delta^{O(1)}\right)\right)$ expected time, where $\Delta$ is the diameter, simply by setting $\varepsilon$ near $1/\Delta$. If one wants running time near linear in $n$, the best previous result we are aware of was by Eppstein [6] and had exponential dependence in $\Delta$ (namely, the time bound is $O\left(n2^{\Delta \log \Delta}\right)$). Note that our result beats Cabello's or Gawrychowski et al.'s algorithm when the diameter is smaller than $n^\delta$ for some constant $\delta$.

## References

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

2 Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *Proceedings of the Tenth International Workshop on Algorithms and Data Structures*, pages 541–552. Springer, 2007.

3 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *Proceedings of the Twenty-eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2143–2152, 2017.

4 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *CoRR*, abs/1702.07815v1, 2017.

5 Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8:1–17, 2012.

6 David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *SODA*, volume 95, pages 632–640, 1995.

7 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

8 Paweł Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. *CoRR*, abs/1704.02793v1, 2017.

9 Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\varepsilon)$-approximate distance oracle for planar graphs. In *Proceedings of the Twenty-sixth International Symposium on Algorithms and Computation*, pages 625–636. Springer, 2015.

10 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

11 David Hartvigsen and Russell Mardon. The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. *SIAM Journal on Discrete Mathematics*, 7(3):403–418, 1994.

12 Monika R. Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

13 Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the Thirty-eight International Colloquium on Automata, Languages, and Programming*, pages 135–146. Springer, 2011.

14 Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 550–563, 2013.

15 Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 820–827, 2002.

16 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 146–155, 2005.

17 Rolf Klein. *Concrete and abstract Voronoi diagrams*, volume 400. Springer Science & Business Media, 1989.

18 Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract Voronoi diagrams. In *Informatik*, pages 283–308. Springer, 1992.

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

20 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

21 Oren Weimann and Raphael Yuster. Approximating the diameter of planar graphs in near linear time. *ACM Transactions on Algorithms*, 12(1):1–13, 2016.

22 Christian Wulff-Nilsen. *Algorithms for planar graphs and graphs in metric spaces.* PhD thesis, PhD thesis, University of Copenhagen, 2010.

## A Approximate Distance Oracles

To construct an approximate distance oracle, we build upon the general framework of the oracles of Thorup, Kawarabayashi et al., and Gu and Xu ([20, 14] and [9] respectively). Given a weighted, undirected planar graph $\mathcal{G}$ with non-negative edge lengths, we will focus on constructing a distance oracle with additive stretch $\varepsilon\Delta$ (also called *additive distance oracle*), where $\Delta$ is the diameter of $\mathcal{G}$. Such an oracle returns, given two vertices $u$ and $v$ of $\mathcal{G}$, an approximation $\hat{d}$ of their distance $d_{\mathcal{G}}(u,v)$ in $\mathcal{G}$, such that $d_{\mathcal{G}}(u,v) \leq \hat{d} \leq d_{\mathcal{G}}(u,v)+\varepsilon\Delta$. A known *scaling* technique (see Kawarabayashi et al. [13] and Section 4 in [9]) can convert the additive distance oracle to a $(1+\varepsilon)$-approximate distance oracle.

To construct the additive distance oracle, we recursively decompose the given graph $\mathcal{G}$ as in Sections 3.1 and 3.4, but here we also store the graphs in a tree, called the *recursive decomposition tree*. Let $N$ be the size of $\mathcal{G}$. Let $\mu$ be an internal node of the recursive decomposition tree, $G^{(\mu)}$ its graph, and $n = |V\left(G^{(\mu)}\right)|$. In the root $\nu$ of the tree, $G^{(\nu)} = \mathcal{G}$. Let $C^{(\mu)}$ be the shortest path separator used to decompose $G^{(\mu)}$ into two disjoint and connected planar subgraphs $G^{(\mu)}_{\text{in}}$ and $G^{(\mu)}_{\text{out}}$ as described in Section 3.1. We find a set $Y^{(\mu)}$ of $O(1/\varepsilon)$ vertices on $C^{(\mu)}$, called *portals*, such that we can approximate *any* shortest path between any $u \in V\left(G^{(\mu)}_{\text{in}}\right)$ and $v \in V\left(G^{(\mu)}_{\text{out}}\right)$ by routing it through one of these portals. Let $\widetilde{\Delta}$ be a 2-approximation of the diameter of $\mathcal{G}$, computed as in 3.1. To find the portals we use Lemma 10, slightly changed for the current setting.

▶ **Lemma 15.** *We can select a set $Y^{(\mu)}$ of $O(1/\varepsilon)$ vertices, where $\varepsilon > 0$, on $C^{(\mu)}$ in linear time, such that $d_{G^{(\mu)}}(u,v) \leq \min_{y\in Y^{(\mu)}}\{d_{G^{(\mu)}}(u,y) + d_{G^{(\mu)}}(y,v)\} \leq d_{G^{(\mu)}}(u,v) + 2\varepsilon\widetilde{\Delta}$ for any $u \in V\left(G^{(\mu)}_{\text{in}}\right)$ and $v \in V\left(G^{(\mu)}_{\text{out}}\right)$.*

We prove the following theorem (similar to Theorem 3), which is crucial into substituting the exponential dependency on $1/\varepsilon$ in the space and the preprocessing time of the additive distance oracle in [9] with a polynomial one, while retaining the constant query time.

▶ **Theorem 16.** *Let $H$ be a weighted, undirected planar graph of $n$ vertices with non-negative edge lengths and $W$ be an integer. Let $X$ be a set of $b$ vertices on the boundary of the outer face of $H$. Let $H^+$ be the graph obtained by adding to $H$ a vertex $z_0$, with an edge from $z_0$ to each vertex $x \in X$ of unspecified length. Let there be $O(n)$ different $b$-tuples of lengths for those edges.*

*We can preprocess $H$ in $O\left(nb^3W^2 + b^4W^2\right)$ expected preprocessing time and space, such that the following query can be answered in $O(1)$ time: given an $O(\log n)$-bit identifier of one of those tuples and a vertex $u \in V(H)$, return $d_{H^+}(z_0,u)$.*

**Proof.** We create a set $S$ of $b$ sites where each site is placed on a different vertex of $X$. For each pair of sites we construct all the different bisectors by using Lemma 5. There are $O(W)$ bisectors for each pair of sites (Lemma 5), so there are $O\left(b^2W\right)$ bisectors in total. For each bisector and each vertex we store a boolean flag which is set to true if the vertex is enclosed by the bisector and false otherwise (that can be done by a variant of BFS) in $O\left(nb^2W\right)$ time. We find all the $O\left(b^4W^2\right)$ pieces of the graph into which it is decomposed by all the bisectors in that much time. The boundary of each such piece is the concatenation of at most $b$ bisectors. For each vertex of $H$ we find in $O(b)$ time the piece that it belongs to and store a pointer to it in $O(nb)$ total time.

For each tuple of lengths we construct the abstract Voronoi diagram of $S$ in $O\left(nb^3W^2\right)$ time, by using Lemma 8, find in $O\left(b^4W^2\right)$ time the Voronoi region enclosing each piece of the previous paragraph, and create a pointer to it. We store the pointer of each piece in a hash table using the identifier of each tuple, but we do not store any abstract Voronoi diagram. The total preprocessing time is $O\left(nb^3W^2 + b^4W^2\right)$ expected. The space required is $O\left(n + b^4W^2\right)$.

In a query, given a $O(\log n)$-bits identifier representing a tuple of lengths and a vertex $u \in V(H)$, we find the piece containing $u$, by using $u$'s pointer, and then query the hash table of that piece to find the site $s$, such that $v_s = \arg\min_{t\in S}\{d_H(v_t, u) + w_t\}$ by using the given identifier as key. Then, we return $d_H(v_s, u) + w_s$, which is $d_{H^+}(z_0, u)$. The query time is constant. ◀

We run an SSSP algorithm from the portals of every internal node $\mu$ of the recursive decomposition tree. Let $\ell$ be the largest distance found. We construct the data structure of Theorem 16 for $H = G_{\text{in}}^{(\mu)}$, after dividing the length of every edge therein by $\varepsilon\ell$, $X = Y^{(\mu)}$, $b = O(1/\varepsilon)$, and $W = 1/\varepsilon$. Each of the $n$ tuples of lengths corresponds to the tuple of the shortest path distances of a different vertex of $G_{\text{out}}^{(\mu)}$, after rounding each to the closest multiple of $\varepsilon\ell$ and dividing by that number, to the portals. Each such tuple is provided with a unique $O(\log n)$-bits identifier.

We create graphs $G_{\text{in}}^{(\mu)+}$ and $G_{\text{out}}^{(\mu)+}$, where $\varepsilon' = \varepsilon/n^{1/8}$, in $O\left(n\log n + (1/\varepsilon')^4 \log n\right)$ time, as in Section 3.4, assign them to the children of $\mu$, and recurse. We stop when the size of the graph is $O(1/\varepsilon)$. The height of the recursive decomposition tree is $O(\log n)$. For each leaf node of the tree we run a brute-force APSP algorithm and store the distance matrix. We also preprocess the tree as in [10], such that we can answer lowest common ancestor queries in $O(1)$ time.

To answer a query, given two vertices $u$ and $v$, let $\mu_u$ and $\mu_v$ respectively be the nodes of the recursive decomposition tree containing it. If $\mu_u = \mu_v$ and $\mu_u$ is a leaf, we return the shortest path distance from $u$ to $v$ by visiting the distance matrix therein. Else, we find in $O(1)$ time their lowest common ancestor $\mu_{u,v}$; supposing w.l.o.g. that $u \in V\left(G_{\text{in}}^{(\mu_{u,v})}\right)$ and $v \in V\left(G_{\text{out}}^{(\mu_{u,v})}\right)$, we properly query the data structure of Theorem 16 of $\mu_{u,v}$, and return the distance found, multiplied with $\varepsilon\ell$. Similarly to Lemma 13 we have the following lemma.

▶ **Lemma 17.** *For two vertices $u, v \in V(\mathcal{G})$ the additive oracle returns an value $\hat{d}$ such that $d_{\mathcal{G}}(u,v) \le \hat{d} \le d_{\mathcal{G}}(u,v) + O(\varepsilon)\Delta$.*

Finally, we bound the query time, the space, and the preprocessing time of our additive distance oracle.

▶ **Theorem 18.** *The space occupied by the additive oracle is $O\left(N\left(\log N + (1/\varepsilon)^6\right)\right)$, the preprocessing time required is $O\left(N\left(\log N\left(\log N + (1/\varepsilon)^5\right) + (1/\varepsilon)^6\right)\right)$, and a query can be answered in $O(1)$ time.*

**Proof.** It takes $O(1)$ time to find the lowest common ancestor of two nodes and to query the distance of any vertices therein. It also takes $O(1)$ time to query the distance matrix in a leaf. Therefore, the query time is $O(1)$.

The preprocessing time $T(n)$ and space $S(n)$ satisfy the following recurrence relations:

$$\begin{aligned}
T(n) \quad &\le \max_{1/3\le\alpha\le 2/3} \left(T\left(\alpha n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + T\left((1-\alpha)n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + \right.\\
&\qquad \left. O\left(n\left(\log n + (1/\varepsilon)^5\right) + (1/\varepsilon)^6\right)\right),\\
S(n) \quad &\le \max_{1/3\le\alpha\le 2/3} \left(S\left(\alpha n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + S\left(\beta n + O\left((1/\varepsilon)^4\sqrt{n}\right)\right) + O\left(n + (1/\varepsilon)^6\right)\right).
\end{aligned}$$

In the base case $n = O\left(1/\varepsilon^4\right)$, so we run a quadratic-time APSP algorithm in $O\left(n^2\right)$ time. Since we have $O\left(\varepsilon^4 N\right)$ such graphs, the total time for the base case is $O\left((1/\varepsilon)^4 N\right)$. The solutions to the recurrences are $T(N) = O\left(N\left(\log N\left(\log N + (1/\varepsilon)^5\right) + (1/\varepsilon)^6\right)\right)$ and $S(N) = O\left(N\left(\log N + (1/\varepsilon)^6\right)\right)$. ◀