

Faster Approximate Diameter and Distance Oracles in Planar Graphs

Timothy M. Chan¹ and Dimitrios Skrepetos²

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

² Cheriton School of Computer Science, University of Waterloo
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 $O(n \log n (\log n + (1/\varepsilon)^5))$ expected time, improving upon the $O(n((1/\varepsilon)^4 \log^4 n + 2^{O(1/\varepsilon)}))$ -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.2017.1](https://doi.org/10.4230/LIPIcs.ESA.2017.1)

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 lengths¹, defined as the longest shortest path distance between two vertices of the graph. Since Frederickson in 1983 [7] solved the problem in $O(n^2)$ 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 $O(n^2 \log \log n / \log n)$ time; the algorithm of the latter requires the same amount of time for the unweighted case and $O(n^2 (\log \log n)^4 / \log n)$ time for the weighted. However, a truly subquadratic algorithm, i.e., an algorithm running in $O(n^{2-\delta})$ 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, Δ , of a graph is a value $\tilde{\Delta}$ such that $\Delta \leq \tilde{\Delta} \leq c\Delta$. Using the linear-time SSSP algorithm of Henzinger et

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.



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 $O(n^{3/2})$ 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 $O(n((1/\varepsilon)^4 \log^4 n + 2^{O(1/\varepsilon)}))$. 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 $\tilde{O}(n^{11/6})$ 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 $\tilde{O}(n^{5/3})$ worst-case time. No lower bound is available presently, but Cabello [4] conjectured that the diameter cannot be computed exactly in time faster than $O(n^{1+\delta})$, 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 $O(n \log n (\log n + (1/\varepsilon)^5))$ 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 $d_G(u, v) \leq \tilde{d} \leq (1+\varepsilon)d_G(u, v)$, 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 $O((1/\varepsilon)^2 n \log^3 n)$ preprocessing time, $O((1/\varepsilon)n \log n)$ 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 ε); it requires $O(n \log n ((1/\varepsilon)^2 \log^3 n + 2^{O(1/\varepsilon)}))$ preprocessing time and $O(n \log n ((1/\varepsilon) \log n + 2^{O(1/\varepsilon)}))$ 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*

86 $O(1)$ query time, $O(n \log n (\log n (\log n + (1/\varepsilon)^5) + (1/\varepsilon)^6))$ expected preprocessing time,
 87 and $O(n \log n (\log n + (1/\varepsilon)^6))$ space.

88 Throughout the paper we operate under the standard RAM model of computation. Let
 89 $[W] = \{1, \dots, W\}$. We assume that all the planar graphs under discussion have a fixed,
 90 combinatorial embedding and are triangulated.

91 **2 A Streamlined Version of Cabello's Technique**

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

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

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

101 In our application, $b = W = O(1/\varepsilon)$, so the preprocessing time would be near linear
 102 in n . Cabello established a similar theorem [4, Theorem 21] for the more general setting
 103 where each edge z_0x ($x \in X$) may have a real length, but his preprocessing time bound is
 104 $\tilde{O}(n^2b^3 + b^4)$. We show that when the length of each edge z_0x ($x \in X$) is a small integer,
 105 the preprocessing time can be greatly improved, and at the same time the method becomes
 106 simpler.

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

110 **2.1 Defining Voronoi diagrams in planar graphs**

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

114 Each *site* s of a Voronoi diagram in a planar graph is a pair (v_s, w_s) , where v_s is the
 115 site's placement, i.e., a vertex of the graph, and w_s is its weight. Given a graph G and a
 116 set of sites S , the *Voronoi region* of a site $s \in S$ is defined as $\text{VR}_G(s, S) = \{u \in V(G) \mid$
 117 $d_G(v_s, u) + w_s \leq d_G(v_t, u) + w_t, \forall t \in S - \{s\}\}$, i.e., as the set of all vertices closer to s
 118 than to any other site under the weighted metric; the *Voronoi diagram* of S is defined as
 119 $\text{VD}_G(S) = \mathbb{R}^2 \setminus \bigcup_{s \in S} \text{VR}_G(s, S)$.

120 A key concept in Voronoi diagrams is bisectors. The bisector of two sites s and t ,
 121 $\text{bis}_G(s, t)$, is defined as the set of the duals of the edges in $E_G(s, t) = \{uv \in E(G) \mid$
 122 $d_G(u, v_s) + w_s \leq d_G(u, v_t) + w_t$ and $d_G(v, v_t) + w_t \leq d_G(v, v_s) + w_s\}$, i.e., the bisector
 123 contains the duals of all the edges whose endpoints are not both closer to the same site.
 124 Let $\{p, q\}$ be a generic (i.e., for each $u \in V(G)$ we have $d_G(u, v_p) + w_p \neq d_G(u, v_q) + w_q$)
 125 and independent (i.e., each Voronoi region is non-empty) set of sites on the boundary of the
 126 *outer face* of a planar graph G . Then the bisector of p and q is a simple cycle in the dual,
 127 passing through the dual vertex, v_∞ , of the outer face ([4, Lemma 5]).

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

134 2.2 Computing abstract Voronoi diagrams in planar graphs

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

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

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

145 We start by showing how to compute all different bisectors, given two vertices of X as
 146 placements of sites, whose weights are drawn from $[W]$, by building upon [4, Lemma 17].
 147 We need $O(nW)$ total time for constructing the bisectors, whereas Cabello needed $O(n^2)$
 148 time for general real weights; we can return a pointer to a bisector in $O(1)$ time instead of
 149 $O(\log n)$ time.

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

155 **Proof.** Assuming w.l.o.g. that $w_s \geq w_t$, we can write $\text{bis}_H((v_s, w_s), (v_t, w_t))$ as
 156 $\text{bis}_H((v_s, w), (v_t, 0))$, where $w = w_s - w_t$, so we need to consider only $\text{bis}_H((v_s, w), (v_t, 0))$,
 157 where $w \in [W]$. Hence, there can be at most $O(W)$ different bisectors for a pair of sites.

158 Let $s = (v_s, w_s), t = (v_t, w_t)$, and $S = \{s, t\}$. For each vertex $u \in V(H)$ we compute
 159 the value $\eta_u = d_H(v_t, u) - d_H(v_s, u)$, by first running the linear-time SSSP algorithm of
 160 Henzinger et al. [12] from v_s and v_t and then visiting each vertex; $u \in V(H)$ belongs to
 161 $\text{VD}_H(s, \{s, t\})$ when $w \leq \eta_u$ and to $\text{VD}_H(t, \{s, t\})$ otherwise. For each $w \in [W]$ we compute
 162 the bisector $\text{bis}_H((v_s, w), (v_t, 0))$, by marking each edge $uv \in E(G)$ such that $w \leq \eta_u$ and
 163 $w > \eta_v$. The bisector is composed of the duals of the marked edges and is a cycle in the dual,
 164 passing through v_∞ ; we represent it as a linked list, $\text{LL}_{s,t,w}$. Finally, we store the linked-list
 165 representation of every different bisector $\text{bis}_H((v_s, w), (v_t, 0))$ in a table, $\text{T}_{s,t}$, indexed by w .

166 The total time spent is $O(nW)$ because we have at most W different bisectors, and
 167 computing each takes $O(n)$ time. Given two sites s and t with weights w_s and $w_t \in [W]$
 168 respectively, we can return a pointer to the pertinent bisector in $O(1)$ time by looking up
 169 $\text{T}_{s,t}[w]$, assuming w.l.o.g. that $w_s \geq w_t$. ◀

170 Next, we show how to compute all different Voronoi diagrams, given three vertices of X
 171 as placements of sites, whose weights are drawn from $[W]$, by building upon [4, Lemma 18].

172 We need $O(nW^2)$ time, whereas Cabello had $O(n^2)$; we can return a pointer to a Voronoi
 173 diagram in $O(1)$ time instead of $O(\log n)$. Furthermore, our proof is simpler since it does
 174 not involve line arrangements and amortization.

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

180 **Proof.** We invoke Lemma 5 to compute and store all the different bisectors of each pair
 181 of the three sites in $O(nW)$ time. We can assume w.l.o.g. that $w_q = 0$, so there are at
 182 most $O(W^2)$ different Voronoi diagrams. Let $s = (v_s, w_s), t = (v_t, w_t), q = (v_q, w_q)$, and
 183 $S = \{s, t, q\}$. For each vertex $u \in V(H)$ we compute the values $\eta_x^{st} = d_H(v_s, u) - d_H(v_t, u)$,
 184 $\eta_x^{qt} = d_H(v_q, u) - d_H(v_t, u)$, and $\eta_x^{sq} = d_H(v_s, u) - d_H(v_q, u)$ by running the SSSP algorithm
 185 of [12]; u belongs to $\text{VR}_H(s, \{s, t, q\})$ if $\eta_u^{st} \leq w_t - w_s$ and $\eta_u^{sq} \leq -w_s$; similar statements
 186 can be made for $\text{VR}_H(t, \{s, t, q\})$ and $\text{VR}_H(q, \{s, t, q\})$.

187 For every $w_s, w_t \in [W]$, we find in linear time the Voronoi diagram $\text{VD}_H(S)$ as follows.
 188 Each bisector (i) does not participate at all, (ii) participates wholly, or (iii) only a subpath
 189 of it, passing through v_∞ , participates in $\text{VD}_H(S)$ (that is implied by that fact that $\text{VD}_H(S)$
 190 has at most one vertex besides v_∞ ; see [4, Lemma 13]). We provide two pointers for each
 191 bisector, which mark the bisector's part that constitutes a Voronoi edge: one for its first and
 192 one for its last edge participating in $\text{VD}_H(S)$. Starting from v_∞ , we scan the edges of each
 193 bisector in clockwise order; the first pointer of $\text{bis}_H(s, t)$ is created for the first encountered
 194 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
 195 (which can be determined using their η values). The second pointer is created for the last
 196 such edge. If no such edges are encountered, both pointers are set to NULL. We can find
 197 in $O(1)$ time if there exists a Voronoi vertex; to do so, we scan each triple of pointers of
 198 the bisectors to see if the corresponding edges meet at a common dual vertex. If that is the
 199 case, we set that vertex to be a Voronoi vertex.

200 The representation of $\text{VD}_H(S)$ for weights $w_s, w_t \in [W]$ is composed of (i) a linked list
 201 of each bisector participating in it, (ii) the first and last pointers of each such bisector, and
 202 (iii) the one, if any, Voronoi vertex therein, besides v_∞ . Each different Voronoi diagram is
 203 stored in a two-dimensional table $T_{s,t,q}$, indexed by w_s and w_t . Given weights w_s, w_t , and
 204 w_q , where w.l.o.g. $w_s, w_t \geq w_q$ we can return a pointer to the pertinent Voronoi diagram
 205 in $O(1)$ time by looking up $T_{s,t,q}[w_s - w_q, w_t - w_q]$, assuming w.l.o.g. that $w_s \geq w_q$ and
 206 $w_t \geq w_q$. ◀

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

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

216 Now we can use Lemma 7 and Theorem 4 to compute the abstract Voronoi diagram of b
 217 sites, given all the vertices of X as placements of sites, whose weights are drawn from $[W]$.

218 Our preprocessing time is $O(nb^3W^2)$ expected, whereas Cabello had $O(n^2b^3)$; our query
219 time is $O(b \log b)$, while Cabello had multiple $\log n$ factors.

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

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

228 2.3 Constructing the farthest neighbor data structure

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

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

238 We can now prove the main theorem of this section.

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

245 In a query, we compute the abstract Voronoi diagram of H , where for each $s \in S$ we
246 set w_s to be equal to the given length of the edge w_0v_s , by using the data structure of
247 Theorem 8. For each site $s \in S$, we query the data structure of Lemma 9 for s to find the
248 vertex of $\text{VR}_H(s, S)$ with the largest distance from s by walking along its boundary, which
249 is the concatenation of at most b subpaths of the bisectors $\text{bis}_H(s, \cdot)$. From Lemma 9 we
250 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
251 distances. Thus, the total query time is $O(b \log b)$. ◀

252 3 Improving Weimann and Yuster's Diameter Approximation 253 Algorithm

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

256 Let \mathcal{G} be the original graph and N its size. Let $d(G_1, G_2, G_3)$ denote the longest shortest
257 path distance between a marked vertex of G_1 and a marked vertex of G_2 in G_3 . Initially
258 $G = \mathcal{G}$, n is the size of G , all vertices are marked, and we want to approximate $d(G, G, G)$.
259 Let $\varepsilon > 0$. The outline of the recursive scheme is as follows.

- 260 1. Find a cycle C of G , such that the removal of C 's vertices decomposes G into two disjoint
 261 and connected planar graphs A and B , each having between $n/3$ and $2n/3$ vertices; C
 262 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
 263 A (resp. B) lies inside (resp. outside) C .
- 264 2. Approximate $d(G_{\text{in}}, G_{\text{out}}, G)$ (Sections 2.1 and 2.2 in [21]).
- 265 3. Unmark all vertices of C and build graphs G_{in}^+ and G_{out}^+ such that (i) they are planar,
 266 and connected graphs, (ii) each of G_{in}^+ and G_{out}^+ has at most roughly $2n/3$ vertices, (iii)
 267 together they have at most roughly n vertices, and (iv) $d(G_{\text{in}}, G_{\text{in}}^+, G_{\text{in}}^+)$ is a $(1 + \varepsilon')$ -
 268 approximation of $d(G_{\text{in}}, G_{\text{in}}, G)$ for an appropriate choice of parameter ε' . Then recurse
 269 in G_{in}^+ to approximate $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+)$ and do the same for G_{out}^+ (Section 2.3 in [21]).
- 270 4. Return $\max \{d(G_{\text{in}}, G_{\text{out}}, G), d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+), d(G_{\text{out}}, G_{\text{out}}, G_{\text{out}}^+)\}$.

271 3.1 Decomposing G to G_{in} and G_{out}

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

282 3.2 Reducing $d(G_{\text{in}}, G_{\text{out}}, G)$ to $d(G_{\text{in}}, G_{\text{out}}, G_{\text{p}})$

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

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

292 We run an SSSP algorithm from each portal of Y in G ; let ℓ be the largest distance
 293 found. We construct a graph $G_{\text{p}} = G_{\text{p},\text{in}} \cup G_{\text{p},\text{out}}$, such that $V(G_{\text{p},\text{in}}) = V(A) \cup Y$ and
 294 $V(G_{\text{p},\text{out}}) = Y \cup V(B)$. We create an edge between each vertex of $G_{\text{p},\text{out}}$ and each portal,
 295 whose length is equal to their shortest path distance in G , after rounding it to the closest
 296 multiple of $\varepsilon\ell$ and dividing it by that number. The edges between vertices of $G_{\text{p},\text{in}}$ are the
 297 same as in G , but their lengths are also divided by $\varepsilon\ell$. The total time for the reduction is
 298 $O((1/\varepsilon)n)$.

299 3.3 Approximating $d(G_{\text{in}}, G_{\text{out}}, G_{\text{p}})$

300 We construct the farthest neighbor data structure of Theorem 3 for $H = G_{\text{in}}$, $X = Y$,
 301 $b = O(1/\varepsilon)$, and $W = 1/\varepsilon$. Then, we query it n times, by using each vertex $u \in V(G_{\text{out}})$

302 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
 303 the vertices of $V(G_{\text{in}})$. Finally, we return the maximum of the distances found, multiplied
 304 by $\varepsilon\ell$. The total time for approximating $d(G_{\text{in}}, G_{\text{out}}, G_p)$ is thus $O(nb^3W^2 + nb \log b) =$
 305 $O((1/\varepsilon)^5n)$.

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

310 3.4 Reducing $d(G_{\text{in}}, G_{\text{in}}, G)$ to $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+)$

311 After approximating $d(G_{\text{in}}, G_{\text{out}}, G)$, we need to approximate $d(G_{\text{in}}, G_{\text{in}}, G)$ and
 312 $d(G_{\text{out}}, G_{\text{out}}, G)$; however, we cannot directly recurse in G_{in} and G_{out} respectively because
 313 two problems arise (since the treatment is symmetrical for both G_{in} and G_{out} , we concern
 314 ourselves only with the former). First, a path realizing $d(G_{\text{in}}, G_{\text{in}}, G)$ could have a subpath
 315 lying in G_{out} . Second, G_{in} can have up to $O(n)$ vertices because C , which is part of G_{in} ,
 316 itself could have that many. However, G_{in} has at most $2n/3$ marked vertices, which are the
 317 only ones used for computing $d(G_{\text{in}}, G_{\text{in}}, G)$. Therefore, we need to construct graphs G_{in}^+
 318 and G_{out}^+ such that (i) they are planar, and connected graphs, (ii) each of G_{in}^+ and G_{out}^+ has
 319 at most roughly $2n/3$ vertices, (iii) together they have at most roughly n vertices, and (iv)
 320 $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+)$ is a $(1 + \varepsilon')$ -approximation of $d(G_{\text{in}}, G_{\text{in}}, G)$ for an appropriate choice of
 321 parameter ε' . The following lemma is from [21, Lemma 2.3].

322 ► **Lemma 11.** *If $d(G_{\text{in}}, G_{\text{in}}, G) \geq \tilde{\Delta}$, then $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+)$ is a $(1 + 2\varepsilon')$ -approximation of*
 323 *$d(G_{\text{in}}, G_{\text{in}}, G)$. Otherwise, $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}}^+) \leq (1 + 2\varepsilon')\tilde{\Delta}$.*

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

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

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

345 **Proof.** We apply the multiple-source shortest paths data structure of Klein, which preprocesses
 346 a planar graph F of n vertices in $O(n \log n)$ time, such that given a vertex u on the

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

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

363 We find p_1 by performing a binary search on the a -to- b shortest path as follows. Let p'
 364 and p'' be initially set to a and b respectively. Let p be the vertex midway between p' and p''
 365 on the a -to- b path, which can be found by a level ancestor query. We want to find whether
 366 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
 367 p_2 and b . To do so, we find the lowest common ancestor lca of p and d on the shortest path
 368 tree of c . If $lca = p$, then we are in case (i). Else, we perform a level ancestor query for p and
 369 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
 370 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
 371 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)
 372 or (iii) we recurse with $p'' = p$; for case (ii) we recurse with $p' = p$. We stop when $p' = p''$.

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

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

385 3.5 Analyzing the approximation factor and the running time

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

387 **Proof.** Let $G^{(\mu)}$ be the graph of a node μ of the recursion tree, and $G_{\text{in}}^{(\mu)}$ and $G_{\text{out}}^{(\mu)}$ be
 388 the two graphs created by decomposing it, as in Section 3.1. We $(1 + O(\varepsilon))$ -approximate
 389 $d(G_{\text{in}}^{(\mu)}, G_{\text{out}}^{(\mu)}, G^{(\mu)})$ for each node μ of the recursion tree, so the approximation factor of

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

our algorithm is $(1 + O(\varepsilon)) \max_{\mu} \frac{d(G_{\text{in}}^{(\mu)}, G_{\text{out}}^{(\mu)}, G^{(\mu)})}{d(G_{\text{in}}^{(\mu)}, G_{\text{out}}^{(\mu)}, \mathcal{G})} \leq (1 + O(\varepsilon)) \prod_i (1 + \varepsilon_i)$ where $\varepsilon_i = \varepsilon/n_i^{1/8}$,
 for some sequence n_1, n_2, \dots, n_k satisfying $n_{i-1}/3 + \Theta((1/\varepsilon_i)^4) \leq n_i \leq 2n_{i-1}/3 + \Theta((1/\varepsilon_i)^4)$
 with $n_1 = N$ and $n_k = O((1/\varepsilon)^4)$.

Now, $\prod_i (1 + \varepsilon_i) \leq \exp\left(\sum_i \varepsilon_i\right)$. Since n_i decreases at least exponentially, ε_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 ε by a constant
 factor). \blacktriangleleft

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

Proof. The running time satisfies the following recurrence relation:

$$T(n) \leq \max_{1/3 \leq \alpha \leq 2/3} (T(\alpha n + O((1/\varepsilon)^4 \sqrt{n})) + T((1 - \alpha)n + O((1/\varepsilon)^4 \sqrt{n})) + O(n(\log n + (1/\varepsilon)^5))).$$

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

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 re-
 quires $\tilde{O}(n^{5/3})$ 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 $O(n \log n (\log n + \Delta^{O(1)}))$ expected time, where Δ is the
 diameter, simply by setting ε 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 Δ (namely, the time bound is $O(n 2^{\Delta \log \Delta})$). Note that our result beats Cabello's or
 Gawrychowski et al.'s algorithm when the diameter is smaller than n^δ for some constant δ .

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.

- 425 **3** Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances
426 in planar graphs. In *Proceedings of the Twenty-eighth Annual ACM-SIAM Symposium on*
427 *Discrete Algorithms*, pages 2143–2152, 2017.
- 428 **4** Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances
429 in planar graphs. *CoRR*, abs/1702.07815v1, 2017.
- 430 **5** Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$
431 time. *ACM Transactions on Algorithms*, 8:1–17, 2012.
- 432 **6** David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *SODA*,
433 volume 95, pages 632–640, 1995.
- 434 **7** Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applica-
435 tions. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.
- 436 **8** Paweł Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann.
437 Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$
438 time. *CoRR*, abs/1704.02793v1, 2017.
- 439 **9** Qian-Ping Gu and Gengchun Xu. Constant query time $(1 + \varepsilon)$ -approximate distance or-
440 acle for planar graphs. In *Proceedings of the Twenty-sixth International Symposium on*
441 *Algorithms and Computation*, pages 625–636. Springer, 2015.
- 442 **10** Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors.
443 *SIAM Journal on Computing*, 13(2):338–355, 1984.
- 444 **11** David Hartvigsen and Russell Mardon. The all-pairs min cut problem and the minimum
445 cycle basis problem on planar graphs. *SIAM Journal on Discrete Mathematics*, 7(3):403–
446 418, 1994.
- 447 **12** Monika R. Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster
448 shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*,
449 55(1):3–23, 1997.
- 450 **13** Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate
451 distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the*
452 *Thirty-eight International Colloquium on Automata, Languages, and Programming*, pages
453 135–146. Springer, 2011.
- 454 **14** Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles
455 for approximate distances in undirected planar graphs. In *Proceedings of the Twenty-fourth*
456 *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 550–563, 2013.
- 457 **15** Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate
458 distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Dis-*
459 *crete Algorithms*, pages 820–827, 2002.
- 460 **16** Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the*
461 *Sixteenth Annual Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 146–155,
462 2005.
- 463 **17** Rolf Klein. *Concrete and abstract Voronoi diagrams*, volume 400. Springer Science &
464 Business Media, 1989.
- 465 **18** Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of
466 abstract Voronoi diagrams. In *Informatik*, pages 283–308. Springer, 1992.
- 467 **19** Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM*
468 *Journal on Applied Mathematics*, 36(2):177–189, 1979.
- 469 **20** Mikkel Thorup. Compact oracles for reachability and approximate distances in planar
470 digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.
- 471 **21** Oren Weimann and Raphael Yuster. Approximating the diameter of planar graphs in near
472 linear time. *ACM Transactions on Algorithms*, 12(1):1–13, 2016.
- 473 **22** Christian Wulff-Nilsen. *Algorithms for planar graphs and graphs in metric spaces*. PhD
474 thesis, PhD thesis, University of Copenhagen, 2010.

475 **A** Approximate Distance Oracles

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

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

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

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

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

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

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

517 For each tuple of lengths we construct the abstract Voronoi diagram of S in $O(nb^3W^2)$
 518 time, by using Lemma 8, find in $O(b^4W^2)$ time the Voronoi region enclosing each piece of
 519 the previous paragraph, and create a pointer to it. We store the pointer of each piece in

520 a hash table using the identifier of each tuple, but we do not store any abstract Voronoi
 521 diagram. The total preprocessing time is $O(nb^3W^2 + b^4W^2)$ expected. The space required
 522 is $O(n + b^4W^2)$.

523 In a query, given a $O(\log n)$ -bits identifier representing a tuple of lengths and a vertex
 524 $u \in V(H)$, we find the piece containing u , by using u 's pointer, and then query the hash
 525 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
 526 the given identifier as key. Then, we return $d_H(v_s, u) + w_s$, which is $d_{H^+}(z_0, u)$. The query
 527 time is constant. \blacktriangleleft

528 We run an SSSP algorithm from the portals of every internal node μ of the recursive
 529 decomposition tree. Let ℓ be the largest distance found. We construct the data structure of
 530 Theorem 16 for $H = G_{\text{in}}^{(\mu)}$, after dividing the length of every edge therein by $\varepsilon\ell$, $X = Y^{(\mu)}$,
 531 $b = O(1/\varepsilon)$, and $W = 1/\varepsilon$. Each of the n tuples of lengths corresponds to the tuple of
 532 the shortest path distances of a different vertex of $G_{\text{out}}^{(\mu)}$, after rounding each to the closest
 533 multiple of $\varepsilon\ell$ and dividing by that number, to the portals. Each such tuple is provided with
 534 a unique $O(\log n)$ -bits identifier.

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

541 To answer a query, given two vertices u and v , let μ_u and μ_v respectively be the nodes
 542 of the recursive decomposition tree containing it. If $\mu_u = \mu_v$ and μ_u is a leaf, we return the
 543 shortest path distance from u to v by visiting the distance matrix therein. Else, we find in
 544 $O(1)$ time their lowest common ancestor $\mu_{u,v}$; supposing w.l.o.g. that $u \in V(G_{\text{in}}^{(\mu_{u,v})})$ and
 545 $v \in V(G_{\text{out}}^{(\mu_{u,v})})$, we properly query the data structure of Theorem 16 of $\mu_{u,v}$, and return
 546 the distance found, multiplied with $\varepsilon\ell$. Similarly to Lemma 13 we have the following lemma.

547 **► Lemma 17.** *For two vertices $u, v \in V(\mathcal{G})$ the additive oracle returns an value \hat{d} such that*
 548 $d_{\mathcal{G}}(u, v) \leq \hat{d} \leq d_{\mathcal{G}}(u, v) + O(\varepsilon)\Delta$.

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

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

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

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

$$T(n) \leq \max_{1/3 \leq \alpha \leq 2/3} (T(\alpha n + O((1/\varepsilon)^4 \sqrt{n})) + T((1-\alpha)n + O((1/\varepsilon)^4 \sqrt{n})) + \\ O(n(\log n + (1/\varepsilon)^5) + (1/\varepsilon)^6)),$$

$$S(n) \leq \max_{1/3 \leq \alpha \leq 2/3} (S(\alpha n + O((1/\varepsilon)^4 \sqrt{n})) + S(\beta n + O((1/\varepsilon)^4 \sqrt{n})) + O(n + (1/\varepsilon)^6)).$$

558 In the base case $n = O(1/\varepsilon^4)$, so we run a quadratic-time APSP algorithm in $O(n^2)$
 559 time. Since we have $O(\varepsilon^4 N)$ such graphs, the total time for the base case is $O((1/\varepsilon)^4 N)$.

1:14 Faster Approximate Diameter and Distance Oracles in Planar Graphs

⁵⁶⁰ The solutions to the recurrences are $T(N) = O(N(\log N(\log N + (1/\varepsilon)^5) + (1/\varepsilon)^6))$ and
⁵⁶¹ $S(N) = O(N(\log N + (1/\varepsilon)^6))$. ◀