MANY DISTANCES IN PLANAR GRAPHS

Sergio Cabello

Many distances in planar graphs[*]

Sergio Cabello[†]

May 14, 2009

Abstract

We show how to compute in $O(n^{4/3} \log^{1/3} n + n^{2/3}k^{2/3} \log n)$ time the distance between k given pairs of vertices of a planar graph G with n vertices. This improves previous results whenever $(n/\log n)^{5/6} \leq k \leq n^2/\log^6 n$. As an application, we speed up previous algorithms for computing the dilation of geometric planar graphs.

1 Introduction

Let $G = (V, E, \ell)$ be a graph, where $\ell : E \to \mathbb{R}_+$ assigns an abstract non-negative edge-length[1] to each edge. The edge-lengths define a distance $d_G(\cdot, \cdot)$ between the vertices of the graph, where $d_G(u, v)$ is defined as the minimum of the lengths of the paths connecting the vertices u, v. Consider the following natural problem.

> *k-many distances*
> Given a graph G with abstract edge-lengths and k pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, find the distances $d_G(s_1, t_1), \ldots, d_G(s_k, t_k)$.

In this work, we present new algorithms and data structures for solving the k-many distances problem in planar graphs. Henceforth, we confine the discussion to planar graphs, and use n to denote the number of vertices. An appealing version of the k-many distances problem is when $k = n$, since in this case each vertex may participate in some pair. For this version of the problem, we reduce the previously known running time from $O(n^{3/2})$ to $O(n^{4/3} \log n)$ time. In general, we show how to solve the k-many distances problem in $O(n^{4/3} \log^{1/3} n + n^{2/3}k^{2/3} \log n)$ time, which improves previous results for a large range of values k. (See the discussion below.) Our results rely on topological properties and the existence of cycle-separators in planar graphs.

Like previous approaches, we construct a data structure that can answer queries concerning the distance between any pair of vertices, and then repeatedly query the data structure with the k pairs. For many queries, our data structure has a better trade-off between construction and query time that previous ones. Using rebuilding techniques for the data structure, we can also solve in the same time bound the k-many distances problem when the pairs are not known in advance and the value k is unknown beforehand. On the other

---

[*]A preliminary version was presented at the 17th Annual ACM-SIAM Symposium on Discrete algorithms [2].

[†]Department of Mathematics, Faculty of Mathematics and Physics, University of Ljubljana, Slovenia, and Department of Mathematics, Institute for Mathematics, Physics and Mechanics, Slovenia. `sergio.cabello@fmf.uni-lj.si`. Partially supported by the European Community Sixth Framework Programme under a Marie Curie Intra-European Fellowship and by the Slovenian Research Agency, project J1-7218 and program P1-0297.

[1]Other authors use the term edge-weights.

hand, when the k pairs are known in advance, we can avoid constructing the data structure explicitly and reduce the space that is used.

We also present as an application of our results how to speed up previous results to compute the stretch factor of a geometric planar graph. In our previous version of this work [2], we also discussed improvements on previous algorithms for finding a shortest non-contractible cycle in a graph embedded in a surface of bounded genus, orientable or not. For orientable surfaces, these results have been already improved [3, 16] using different techniques. The full version of [3] shall also treat non-orientable surfaces.

1.1 Our results and roadmap

Let G be a planar graph with n vertices and non-negative edge-lengths. Our main contributions are as follows.

- For any value S in the interval $[n^{4/3} \log^{1/3} n, n^2]$, we construct in $O(S)$ time a data structure of size $O(S)$ that answers distance queries in $O((n/\sqrt{S}) \log^{3/2} n)$ time per query. See Theorem 12.

- The k-many distances problem in G can be solved in $O(k^{2/3} n^{2/3} \log n + n^{4/3} \log^{1/3} n)$ time, even when we do not know the value k or the pairs beforehand. See Theorem 13.

- The k-many distances problem in G can be solved in $O(k^{2/3} n^{2/3} \log n + n^{4/3} \log^{1/3} n)$ time and $O(n + k)$ space when we know the pairs beforehand. See Theorem 14.

- We show the following application: Let G be a planar, Euclidean graph with n vertices in $\mathbb{R}^d$ for some fixed d. Let $3 \geq \varepsilon > 0$ be a constant, and let t_G be the stretch factor of G. We can compute in $O(n^{4/3} \log n)$ time a value t such that $t \leq t_G \leq (1 + \varepsilon)t$.

Model. We use an addition/comparison model of computation, that is, the edge-lengths can be arbitrary real, non-negative values and we only compare values that come from sums of edge-lengths. In the RAM model of computation, some logarithmic improvements may be possible; see Zwick [22] for a discussion. For the last application, computing the stretch factor of geometric graphs, we have to compare sums of square roots. Therefore, we assume a Real RAM model of computation, as it is standard in the context of geometric spanners.

Roadmap. In next subsection we review results concerning distances in graphs and compare them to our results when applicable. In Section 2 we introduce notation and give a toolbox that will be used through the paper. In Section 3 we show how to compute the distances from the boundary of a subgraph to its vertices. In Section 4 we describe the data structure for answering distance queries in planar graphs and analyze it. In Section 5 we describe solutions to the k-many distances problem, and in Section 6 we discuss the application to computing the stretch factor. We finish with a discussion in Section 7.

1.2 Previous results and comparative

We review previous data structures for distances in planar graphs that are relevant for the k-many distances problem and compare it to ours. If a data structure takes T time to be constructed and Q time per distance query, then it can be used to solve the k-many distances problem in $T + kQ$ time; when T and Q depend on a parameter, we choose the parameter to minimize $T + kQ$ as a function of k. We also compare the performance of this approach for the k-many distances with our solution.

- Djidjev [6] uses planar separators to give the following data structures:

- For a parameter $S \in [n^{3/2}, n^2]$ constructs a data structure of size $O(S)$ in $O(S)$ time and answers distance queries in $O(n^2/S)$ time. The case $S = n^{3/2}$ was also described by Arikati et al. [1]. Our data structure is better when $S = o(n^2/\log^3 n)$. Using this data structure, the k-many distances problem can be solved in $O(n^{3/2} + nk^{1/2})$ time. Our approach is better when $k = o(n^2/\log^6 n)$.

  - For a parameter $S \in [n, n^{3/2}]$ constructs a data structure of size $O(S)$ in $O(nS^{1/2})$ time that answers distance queries in $O(n^2/S)$ time. Our data structure is better when $S \geq n^{4/3} \log^{1/3} n$, as otherwise is not defined. Using this data structure, the k-many distances problem can be solved in $O(n^{3/2} + n^{4/3}k^{1/3})$ time if $k \leq n^{5/4}$ or $O(n^{7/4} + kn^{1/2})$ time otherwise. Our result is better for any k.

  - For a parameter $S \in [n^{4/3}, n^{3/2}]$ constructs a data structure of size $O(S)$ in $O(nS^{1/2})$ time and answers distance queries in $O(nS^{-1/2}\log n)$ time. Our data structure answers queries $O(\log^{1/2} n)$ times slower, but it is constructed in $O(S)$ time, which is substantially faster; observe that the ranges of S for both data structures are slightly different. Using this data structure, the k-many distances problem can be solved in $O(n^{5/3} + nk^{1/2}\log^{1/2} n)$ time if $k < n^{3/2}/\log n$ or $O(kn^{1/4}\log n)$ time otherwise. Our approach is better for any k.

- Fakcharoenphol and Rao [8], with the logarithmic improvement by Klein [14], give a data structure that can be constructed in $O(n\log^2 n)$ time and answers distance queries in $O(\sqrt{n}\log^2 n)$ time per query. Using this data structure, the k-many distances problem can be solved in $O(n\log^2 n + k\sqrt{n}\log^2 n)$ time. Our approach is better for $k = \omega((n/\log n)^{5/6})$.

- Henzinger et al. [12] show that a distance in a planar graph can be computed in linear time. Applying this k times gives the best known solution for the k-many distances problem when $k = O(\log^2 n)$.

- Chen and Xu [5] give data structures that depend on a parameter measuring the minimum number of faces over the planar embeddings of the graph, a parameter first introduced by Frederickson [10]. In the worst scenario, this parameter is linear, and for any value $S \in [n^{4/3}, n^2]$, they construct a data structure of size $O(S)$ in $O(n^3/S)$ time if $S \leq n^{3/2}$ and $O(n\sqrt{S})$ time if $S > n^{3/2}$ that answers distance queries in $O(nS^{-1/2}\log(nS^{-1/2}) + \alpha(n^2/S))$ time if $S \geq n^{3/2}$ and $O(S/n\log n)$ time if $S \leq n^{3/2}$. Our data structure has the same query time up to logarithmic factors but has faster construction. This data structure implies that the k-many distances can be solved in $O(n^{3/2} + nk^{1/2}\log^{1/2} n)$. Our approach is better for any k

We conclude that our solution for the k-many distances in planar graphs improves previous results when k satisfies $k = \omega((n/\log n)^{5/6})$ and $k = o(n^2/\log^6 n)$ simultaneously. Our data structure has a better trade-off between construction and query time when many distance queries have to be answered.

For distances in planar graphs with small integer edge-lengths see [15] and references therein. The distance between all pairs of vertices in planar graphs was first solved optimally by Frederickson [10]. For approximate distances in planar graphs, see Thorup [21] and references therein. Finally, our result relies heavily on the recent result by Klein [14] (see also [3]) for solving the many distances problem when all the pairs have a vertex in a common face.
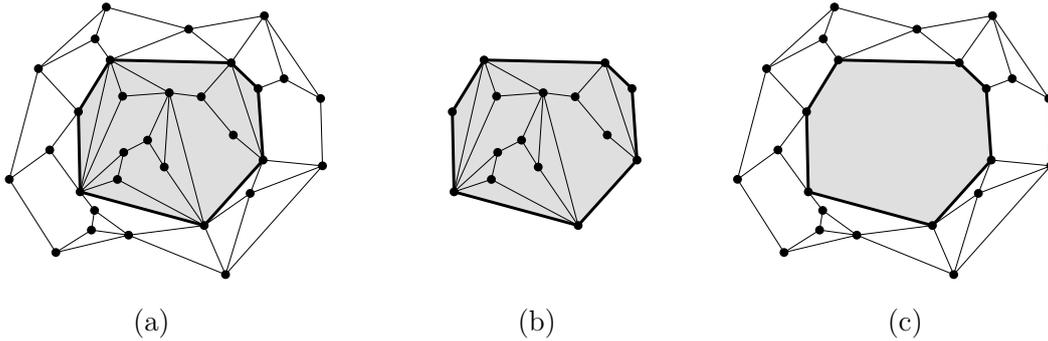
Figure 1: (a) A plane graph B with a cycle C in bold. The gray region is the interior of C. (b) The graph $\texttt{Int}(C, B)$. (c) The graph $\texttt{Ext}(C, B)$.

2 Notation and toolbox

2.1 Basics

Let G be a given planar graph with n vertices. We assume that $V(G) = \{1, \ldots, n\}$ so that arrays can be indexed by the elements of $V(G)$. Each edge $e \in E(G)$ has a non-negative edge length $\ell(e)$. For any set of edges A we use $\ell(A) = \sum_{e \in A} \ell(e)$. For any subgraph B of G, we use $d_B(u, v)$ to denote the length of the shortest path in B between between u and v. When there is no path in B between u and v we write $d_B(u, v) = \infty$.

Using any linear-time embedding algorithm we can assume, henceforth, that G is a *plane* graph, that is, a planar graph together with an embedding in the plane. Using a standard transformation we may further assume that the maximum degree of G is at most three. For non-connected graphs, we will make a slight abuse of terminology and use *facial walk* to refer to the union of the facial walks that define a face that is not simply connected.

We identify a vertex with the point that represents it in the embedding, and an edge with the curve that represents it in the embedding. The term *cycle* is used for a walk with no repeated vertices. A cycle C defines a Jordan curve in the plane, that is, an injective image of $\mathbb{S}^1$. From Jordan's theorem, it follows that $\mathbb{R}^2 \setminus C$ has two connected components, one unbounded, called the *exterior* of C, and one bounded, called the *interior* of C.

Let G' be any embedded planar graph. A cycle C in G' naturally defines two subgraphs; see Figure 1. We use $\texttt{Int}(C, G')$ for the subgraph of G' that is contained in the closure of the interior of C. We use $\texttt{Ext}(C, G')$ for the subgraph of G' that is contained in the closure of the exterior of C. Notice that the edges of C belong to both $\texttt{Int}(C, G')$ and $\texttt{Ext}(C, G')$, while the edges connecting vertices of C that are not part of C go either to $\texttt{Int}(C, G')$ or $\texttt{Ext}(C, G')$, but not to both.

2.2 Pieces

A *piece* B is a subgraph of G; we assume in B the embedding inherited from G. A *boundary walk* of B is a facial walk of B that is not a facial walk of G. A *boundary vertex* of B is a vertex of B incident to an edge in $E(G) \setminus E(B)$. The *boundary* of B, denoted by ∂B, is the set of its boundary vertices. The size of the boundary is the number of vertices in ∂B. Note that a walk in G that intersects B is contained in B or passes through the boundary ∂B.

Let B be a piece and let W be one of its boundary walks. We define the *hole* of B with respect to W, denoted by $H(B, W)$, as the subgraph of G contained in the closure of the face of B defined by W, without the edges of W. See Figure 2. It may be that $H(B, W)$ has several connected components. Since G has maximum degree three, the vertices of W
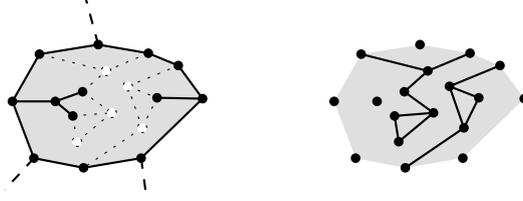
Figure 2: Left: part of a piece B (solid and dashed edges) with a boundary walk W (solid edges) that bounds a face (in light gray). The dotted edges are edges of $E(G) \setminus E(B)$. Right: the hole $H(B, W)$.

are cofacial in $H(B, W)$. (This is not true in general for planar graphs of arbitrary degree because a subtree of W may be enclosed by a cycle of $H(B, W)$.)

If B is a piece with w boundary walks, denoted by $W^1, W^2, \ldots, W^w$, then the edges of the holes $H(B, W^1), H(B, W^2), \ldots, H(B, W^w)$ form a disjoint partition of $E(G) \setminus E(B)$. Each vertex of $V(G)$ that is not in B appears in a unique hole. Given a piece B, we can identify its boundary walks and construct the corresponding holes in linear time. A path in G connecting two vertices of ∂B that is otherwise disjoint from ∂B is contained in B or in one of its holes $H(B, W^j)$.

2.3 Decompositions

The following definition is a variation on the definition by Frederickson [9], where we require that each piece is connected and also consider the number of boundary walks.

Definition 1 *Given a parameter $r \in (0, n)$, an r-decomposition of G consists of a family of pieces $B_1, \ldots, B_p$ such that:*

- *each edge of G appears in at least one piece;*

- *each piece is a connected subgraph;*

- *each piece has at most r vertices;*

- *$p = O(n/r)$, that is, there are $O(n/r)$ pieces;*

- *each boundary ∂B_i has $O(r^{1/2})$ vertices, $i = 1, \ldots, p$;*

- *$\sum_i w_i = O(n/r)$, where w_i is the number of boundary walks of B_i, $i = 1, \ldots, p$.*

We closely follow the approach of Frederickson [9] to give an algorithmic construction of r-decompositions. We first provide a lemma based on the cycle-separator result of Miller [18].

Lemma 2 *Let B be a connected piece with n vertices, b boundary vertices, and w boundary walks. We can construct in linear time two pieces B_1, B_2 such that: $B = B_1 + B_2$, each of the pieces is connected, and both pieces together have $n + O(\sqrt{n})$ vertices, $b + O(\sqrt{n})$ boundary vertices, and $w + 2$ boundary walks. Moreover, we can choose that*

(a) *each piece B_1, B_2 has at most $2n/3 + O(\sqrt{n})$ vertices, or;*

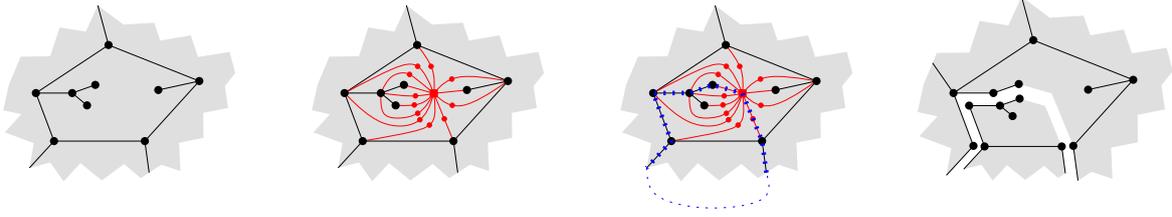(b) *each piece B_1, B_2 has at most $2b/3 + O(\sqrt{n})$ boundary vertices.*

Figure 3: From left to right: a face in a piece B; the face in the supergraph B'; a possible cycle used to split B'; how the split affects the face.

Proof. We construct a supergraph B' of B as follows: for each face f of B we add a new vertex v_f and connect it through a subdivided edge to each appearance of a vertex in the facial walk of f. See Figure 3. The graph B' has a planar embedding that naturally extends the embedding of B: the rotation of the subdivided edges around v_f agrees with the ordering in the facial walk of f. Let V' be the vertices we added to B to create B', that is, $V' = V(B') \setminus V(B)$. The graph B' is 2-connected, has $O(n)$ vertices, and all its faces have 5 vertices.

If we are interested in property (a), we assign weight 1 to each vertex of $V(B)$, and weight 0 to the new vertices V'. For property (b), we would assign weight 1 to each vertex of the boundary of B, and weight 0 to the rest of vertices. We concentrate the rest of our discussion in property (a), since property (b) is handled almost identically. Applying Miller's result [18] we obtain a (simple) cycle C' in B' with $O(\sqrt{n})$ vertices such that each of the subgraphs $\texttt{Ext}(C', B')$ and $\texttt{Int}(C', B')$ contains at most $2n/3 + O(\sqrt{n})$ vertices of $V(B)$.

Let B_1, B_2 be the subpieces $\texttt{Ext}(C', B') - V'$, $\texttt{Int}(C', B') - V'$ of B. Clearly, $B = B_1 + B_2$ and each of the two pieces has at most $2n/3 + O(\sqrt{n})$ vertices. Only the vertices $V(C') \cap V(B)$ go to both B_1 and B_2, and therefore, the total number of vertices and the total number of boundary vertices increases by $O(\sqrt{n})$. The total number of boundary walks in B_1, B_2 is at most $w + 2$: the cycle C' produces two new facial walks (one in B_1 and one in B_2), and a boundary walk of face f in B either disappear (if C' passes through the vertex v_f) or goes to only one subpiece (if C' does not pass through v_f). Finally, note that we extended B to B' in such a way that C' can "go across" a face f of B only once, and the part of f going to either piece is connected, making the pieces B_1, B_2 connected.

The construction of B' from B, the computation of Miller's cycle-separator C', and the remaining steps take linear time. Hence, we use linear time for the whole procedure. □

Theorem 3 *Given a planar graph G with n vertices and a value $r \in (0, n)$, we can construct an r-decomposition of G in $O(n \log(n/r))$ time and $O(n)$ space.*

Proof. Starting with a family of pieces consisting of G, we recursively apply the splitting of Lemma 2, with property (a), to any piece with more than r vertices. Frederickson [9, Lemma 1] argues that this splitting is applied $\Theta(n/r)$ times and, over all pieces, $O(n/\sqrt{r})$ boundary vertices are obtained. The time spent for this part is $O(n \log(n/r))$ because the recursion is balanced and stops when pieces with $\Theta(r)$ vertices are obtained.

Next, we repeatedly apply Lemma 2, with property (b), to each piece of the family that has more than $c\sqrt{r}$ boundary vertices, where $c > 0$ is an appropriate constant. As shown by Frederickson [9, Lemma 2], this splitting takes place $O(n/r)$ times. Eventually we finish with a family of pieces, each with at most r vertices and $O(\sqrt{r})$ boundary vertices. In total, Lemma 2 is used $O(n/r)$ times, and therefore the family we obtain consists of $O(n/r)$ connected pieces and has $O(n/r)$ boundary walks over all pieces. This family is therefore an r-decomposition of G, as sought. Each of the $O(n/r)$ uses of Lemma 2, with property (b), requires $O(r)$ time, and therefore we need $O(n)$ time for this part.

6

At any given time of the algorithm, the sum of the number of vertices in the pieces is bounded by n plus the number of boundary vertices, counted with multiplicity. Once a vertex becomes a boundary vertex in a piece, it keeps being boundary vertex in all the subpieces where it appears. Since there at the end there are $O(n/r)$ pieces with $O(\sqrt{r})$ boundary vertices per piece, the sum of the sizes of the pieces at any given time is bounded by $O(n) + O(n/r) \cdot O(\sqrt{r}) = O(n)$. We conclude that $O(n)$ space is enough to construct the decomposition. □

2.4 Toolbox

We will make use of the following result by Klein [14]; see Cabello and Chambers [3] for an alternative presentation that generalizes to graphs embedded in surfaces.

Theorem 4 *Let G be a plane graph with n vertices, and let F be a face of G. We can preprocess G in $O(n \log n)$ time and space such that any query for a distance $d_G(u, v)$ where u is any vertex of $V(G)$ and v is any vertex of F can be answered in $O(\log n)$ time.*

Under the same hypothesis, and given k pairs of vertices $(u_1, v_1), \ldots, (u_k, v_k)$ with $u_i \in V(G)$ and v_i vertices of F, we can compute the values $d_G(u_1, v_1), \ldots, d_G(u_k, v_k)$ in $O((n + k) \log n)$ time and $O(n + k)$ space.

An *apex graph* G is a graph such that $G - v$ is planar for some vertex $v \in V(G)$; such vertex v is called an *apex* of G.

Lemma 5 *Let G be an apex graph with known apex v. For any vertex $u \in V(G)$, we can compute the values $d_G(u, u')$ for all u' in $V(G)$ in linear time.*

Proof. If G has n vertices, then it has a separator of size $O(\sqrt{n})$: a separator of the planar graph $G - v$ together with v is a separator for G. Moreover, if we know an apex for G, then such separator for G can be found in linear time [17].

Apex graphs are closed under taking subgraphs: a subgraph of an apex graph is an apex graph. Thus, the algorithm by Henzinger et al. [12] imply the result. See Tazari and Müller-Hannemann [20] for an alternative approach. □

It is unclear if this result holds when the apex is unknown because we currently do not known how to find such apex in linear time. We will need to compute the distance between all pairs of vertices in general graphs. The following result is from Chan [4].

Theorem 6 *Given a graph G with n vertices, we can compute the values $d_G(u, u')$ for all pairs of vertices $(u, u') \in V(G) \times V(G)$ in $O(n^3 / \log n)$ time and space.*

Finally, we will also use the following result by Djidjev, which we already mentioned in the introduction.

Theorem 7 *Given a planar graph G with n vertices, there is a data structure of size $O(n^{3/2})$ that can be constructed in $O(n^{3/2})$ time such that a query distance in G can be computed in $O(n^{1/2})$ time.*

In particular, the problem k-many distances can be solved in $O(n^{3/2} + kn^{1/2})$ time and $O(n^{3/2} + k)$ space.

3   Distances within a piece

We first solve a particular problem concerning distances from a piece. Let B be a piece of G with r vertices, w boundary walks, and boundary of size $O(r^{1/2})$.

Lemma 8 *The distances $d_G(u,v)$ for all $u, v \in \partial B$ can be computed in $O(n \log n + r^{3/2})$ time and $O(n + r^{3/2})$ space.*

Proof. Let $\mathbb{W} = \{W^1, \ldots, W^w\}$ be the boundary walks of B. For each $j = 1, \ldots, w$, let H^j denote the hole $H(B, W^j)$. Define a complete graph $K_{\partial B}$ with ∂B as vertex set, and set the length of an edge $uv \in E(K_{\partial B})$ to

$$\min\left(\{d_B(u,v)\} \cup \{d_{H^j}(u,v) \mid u \in W^j \in \mathbb{W} \text{ and } v \in W^j \in \mathbb{W}\}\right).$$

We will see that $d_G(u,v) = d_{K_{\partial B}}(u,v)$ for any $u, v \in \partial B$, and therefore we can compute the desired distances solving the all pairs shortest path problem in $K_{\partial B}$. The rest of the proof is organized as follows. First, we show that indeed $d_G(u,v) = d_{K_{\partial B}}(u,v)$ for any $u, v \in \partial B$. We then bound the running time of the procedure.

To show that $d_G(u,v) = d_{K_{\partial B}}(u,v)$ for any $u, v \in \partial B$, observe first that $d_G(u,v) \leq d_{K_{\partial B}}(u,v)$ because the length of any edge uv in $K_{\partial B}$ is longer $d_G(u,v)$ by construction. To see the other inequality, fix for each pair $u, v \in \partial B$ a shortest path $\Delta_G(u,v)$ in G from u to v that uses the minimum number of vertices from ∂B. We then proceed by induction on the values $|\partial B \cap \Delta_G(u,v)|$, where $u, v \in \partial B$.

- If $|\partial B \cap \Delta_G(u,v)| = 2$, then the path $\Delta_G(u,v)$ is contained in B or in a hole H^j, where W^j is a boundary walk containing u, v. In this case

$$
\begin{aligned}
d_G(u,v) &= \ell(\Delta_G(u,v)) \\
&= \min\{d_B(u,v), d_{H^j}(u,v)\} \\
&\geq d_{K_{\partial B}}(u,v),
\end{aligned}
$$

  which proves the base case of induction.

- If $|\partial B \cap \Delta_G(u,v)| > 2$, consider a vertex v' in $\partial B \cap \Delta_G(u,v)$ different from u, v. Then by the inductive hypothesis we have

$$
\begin{aligned}
d_G(u,v) &= d_G(u,v') + d_G(v',v) \\
&\geq d_{K_{\partial B}}(u,v') + d_{K_{\partial B}}(v',v) \\
&\geq d_{K_{\partial B}}(u,v).
\end{aligned}
$$

This finishes the proof that $d_G(u,v) = d_{K_{\partial B}}(u,v)$ for any $u, v \in \partial B$.

It remains to bound the running time of the procedure. To compute the length of edges $uv \in E(K_{\partial B})$ we proceed as follows.

- The distances $d_B(u,v)$ for all $u, v \in \partial B$ can be computed constructing a shortest path tree from each $u \in \partial B$. Since each such shortest path tree can be constructed in $O(r)$ time because B is planar [12], this takes $|\partial B| \cdot O(r) = O(r^{3/2})$ time and space in total.

- Consider next a fixed $W^j \in \mathbb{W}$. The vertices of W^j are cofacial in H^j, and therefore we can compute the distances $d_{H^j}(u,v)$ for all $u, v \in W^j \cap \partial B$ using Theorem 4. This takes $O((|V(H^j)| + |W^j \cap \partial B|^2) \log n)$ time and $O(|V(H^j)| + |W^j \cap \partial B|^2) = O(n + r)$ space. Repeating this procedure for each boundary walk $W^j \in \mathbb{W}$, we obtain the distances $d_{H^j}(u,v)$ for all $u, v \in W^j \cap \partial B$, $W^j \in \mathbb{W}$. Since G has maximum degree three, each
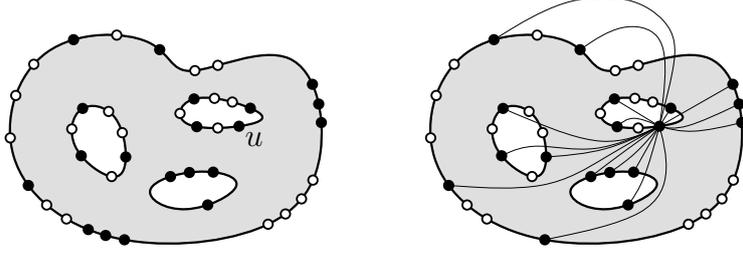
Figure 4: Left: a piece B in light gray; black dots represent boundary vertices and white dots represent non-boundary vertices along the boundary walks. Right: graph B_u for vertex u.

boundary vertex can appear in at most three facial walks and at most three holes. Therefore

$$\sum_j |W^j \cap \partial B| \leq 3|\partial B| = O(r^{1/2})$$

and

$$\sum_j |V(H^j)| \leq 3|V(G)| = O(n).$$

Thus, we spend

$$\sum_j O((|V(H^j)| + |W^j \cap \partial B|^2) \log n) = O\left(\sum_j |V(H^j)| \log n + \sum_j |W^j \cap \partial B|^2 \log n\right)$$

$$= O(n \log n + r \log n)$$

time in this step. Since in each computation involving a walk W^j we can reuse the space, we only need $O(n + r)$ space in this step.

From the distances we have computed we obtain the lengths of the edges in $K_{\partial B}$. The total running time to construct $K_{\partial B}$ is thus $O(r^{3/2} + n \log n + r \log n) = O(n \log n + r^{3/2})$, and we use $O(n + r^{3/2})$ space.

Finally, since $K_{\partial B}$ is a complete graph with $O(r^{1/2})$ vertices, we can compute $d_{K_{\partial B}}(u, v)$ for all vertices $u, v \in \partial B$ in $O(r^{3/2}/\log r) = O(r^{3/2})$ time and space using Theorem 6. □

Lemma 9 *We can compute in $O(n \log n + r^{3/2})$ time and $O(n + r^{3/2})$ space the distances $d_G(u, v)$ for all $u \in \partial B$ and $v \in B$.*

Proof. We use the previous lemma to compute the distances $d_G(u, v)$ for all $u, v \in \partial B$. This takes $O(n \log n + r^{3/2})$ time and uses $O(n + r^{3/2})$ space.

Fix a vertex $u \in \partial B$, and consider the graph B_u obtained by adding to B the additional edges $E_u = \{uv \mid v \in \partial B \setminus \{u\}\}$, each edge $uv \in E_u$ with length $d_G(u, v)$. The graph B_u can be constructed in $O(r)$ time and has size $O(r)$; see Figure 4. Moreover, observe that B_u is an apex graph with known apex vertex u. Using Lemma 5, we can compute the distances $d_{B_u}(u, v)$ for all $v \in B$ in $O(r)$ time and space.

We claim that $d_{B_u}(u, v) = d_G(u, v)$ for any $v \in B$, which implies that we have computed the distances $d_G(u, v)$ for all $v \in B$. Applying this procedure for each of the $O(r^{1/2})$ vertices of ∂B we obtain the lemma.

The proof of the claim is very similar to the one in the previous lemma. Firstly, observe that $d_G(u, v) \leq d_{B_u}(u, v)$ for any vertices $u, v \in B$ because this inequality holds for vertices that are adjacent in B_u. For proving that $d_G(u, v) \geq d_{B_u}(u, v)$ for any vertex $v \in B$,

9

consider a shortest path $\Delta_G(u,v)$ in G from $u \in \partial B$ to $v \in B$. If $|\Delta_G(u,v) \cap \partial B| = 1$, then u is the only boundary vertex of $\Delta_G(v,u)$, which means that $\Delta_G(u,v)$ is contained in B, and therefore $d_G(u,v) = d_B(u,v) = d_{B_u}(u,v)$. If $|\Delta_G(u,v) \cap \partial B| > 1$, let v' be the last vertex in $\Delta_G(u,v) \cap \partial B$ as we walk from u to v; it may be that $v' = v$. We then have $d_G(u,v) = d_G(u,v') + d_G(v',v) = d_{B_u}(u,v') + d_{B_u}(v',v) \geq d_{B_u}(u,v)$. This finishes the proof of the claim and the proof of the lemma. $\qquad\square$

A better time complexity in Lemma 8 can be obtained using the techniques developed by Fakcharoenphol and Rao [8], or by Klein et al. [13]. However, those techniques are more complex and do not improve the time complexity of Lemma 9 or of our forthcoming results.

4 Data structure for distances in planar graphs

Consider a parameter $r \in (0,n)$ whose value will be fixed later. We use Lemma 3 to construct an r-decomposition with pieces $\mathbb{B} = \{B_1, \ldots B_p\}$, where $p = \Theta(n/r)$. We keep using w_i to denote the number of boundary walks of piece $B_i \in \mathbb{B}$. It holds that $\sum_i w_i = O(n/r)$. The data structure $\mathtt{DS}(G,r)$ consists of the following parts:

(a) We store each piece $B_i \in \mathbb{B}$ explicitly with each vertex marked as boundary vertex or non-boundary vertex. We also store with B_i the number $|\partial B_i|$ of boundary vertices, the number w_i of boundary walks, and a linked list with pointers to the boundary vertices ∂B_i.

(b) We store an array $\mathtt{b}[1, \ldots, n]$ of numbers in $\{1, \ldots p\}$ with the property that piece $B_{\mathtt{b}[v]}$ contains vertex v for any $v \in V(G)$.

(c) For each piece $B_i \in \mathbb{B}$ and each boundary walk W_i^j of B_i we store the hole $H_i^j = H(B_i, W_i^j)$ explicitly, and a linked list L_i^j that contains, for each vertex v of $V(H_i^j) \cap (\partial B_i)$, a pointer to the copy of v in H_i^j and a pointer to the copy of v in B_i.

(d) For each piece $B_i \in \mathbb{B}$ we store an array $\mathtt{Locate}_i[1, \ldots, n]$ such that: if vertex $v \in V(G)$ is in B_i, then $\mathtt{Locate}_i(v)$ points to the copy of v in B_i; otherwise $\mathtt{Locate}_i[v]$ points to the copy of v in the unique hole H_i^j that contains v.

(e) For each piece $B_i \in \mathbb{B}$ we store a data structure that reports in $O(1)$ time the distance $d_G(u,v)$ for any query $(u,v) \in (\partial B_i) \times V(B_i)$.

(f) For each piece $B_i \in \mathbb{B}$ we store a data structure that reports in $O(r^{1/2})$ time the distance $d_{B_i}(u,v)$ for any query $(u,v) \in V(B_i)^2$.

(g) For each hole $H_i^j = H(B_i, W_i^j)$ we store a data structure that reports in $O(\log n)$ time the distance $d_{H_i^j}(u,v)$ for any query $(u,v) \in (\partial B_i \cap V(H_i^j)) \times V(H_i^j)$.

We would like to note the discrepancy between the mathematical notation and the representation in data structures. A vertex v of G may appear in several pieces B_i and holes H_i^j, and we do not distinguish them in the mathematical notation. However, we are assuming that each graph stores its own copy of v, and hence it is not the same to access vertex v in the representation of G that in the representation of a piece B_i or a hole H_i^j. However, in our scenario the vertices $V(H_i^j) \cap (\partial B_i)$ will always be accessed through the linked list of part (c), and this gives pointers to both copies of the same vertex.

Lemma 10 *The data structure $\mathtt{DS}(G,r)$ can be constructed in $O((n^2/r) \log n + nr^{1/2})$ time and space.*

Proof. Constructing the r-decomposition takes $O(n \log n)$ time. From the r-decomposition we obtain the pieces $B_1, \ldots, B_p$ explicitly. In $O(n)$ time we can traverse piece B_i in G to identify, mark, and count the boundary vertices of B_i, as well as making the linked list with the boundary vertices. The array $\mathtt{b}[\,]$ of part (b) can be constructed as follows: starting with an empty array $\mathtt{b}[\,]$, we set $\mathtt{b}[v] = i$ for each $v \in V(B_i)$ and each $B_i \in \mathbb{B}$. This sets parts (a) and (b) of $\mathtt{DS}(G, r)$. We have spent $O(n \log n)$ time and $O(n)$ space.

We next describe part (c) of $\mathtt{DS}(G, r)$. For each piece $B_i \in \mathbb{B}$, we proceed as follows. First, we identify the w_i boundary walks $W_i^1, \ldots W_i^{w_i}$ of B_i by traversing B_i in G. Next, for each boundary walk W_i^j of B_i, we construct the hole H_i^j explicitly, taking care to construct the list L_i^j simultaneously: each time we put in H_i^j a vertex v of G that is a boundary vertex of B_i we also include it in L_i^j with the appropriate pointers. In each hole we need time proportional to its number of vertices. Since the holes $H_i^1, \ldots H_i^{w_i}$ are pairwise edge-disjoint, constructing part (c) of $\mathtt{DS}(G, r)$ takes $O(n)$ time per piece $B_i \in \mathbb{B}$.

To construct the array $\mathtt{Locate}_i[\,]$ of part (d) we follow the same approach used for $\mathtt{b}[\,]$. For each piece $B_i \in \mathbb{B}$ we do the following: starting with an empty array $\mathtt{Locate}_i[\,]$, for each v in a hole H_i^j, $j = 1, \ldots w_i$ we set $\mathtt{Locate}_i[v]$ with a pointer to the copy of v in H_i^j. Afterwards, for each $v \in V(B_i)$ we set $\mathtt{Locate}_i[v]$ with a pointer to the copy of v in B_i. This sets part (d) of $\mathtt{DS}(G, r)$. We have spent $O(n)$ time and space per piece $B_i \in \mathbb{B}$.

Part (e) of $\mathtt{DS}(G, r)$ can be constructed as follows. Consider a piece $B_i \in \mathbb{B}$. We attach to each vertex v of B_i a label $\phi_i(v) \in \{1, \ldots, |V(B_i)|\}$ such that the mapping $\phi_i : V(B_i) \to \{1, \ldots, |V(B_i)|\}$ is bijective and $\phi_i(\partial B_i) = \{1, \ldots, |\partial B_i|\}$. That is, ϕ_i is an enumeration of the vertices of B_i where the boundary vertices ∂B_i are at the beginning. Such labeling ϕ_i can be computed easily in $O(r)$ by traversing the piece B_i once to enumerate the vertices of ∂B_i, and then traversing it a second time to enumerate the vertices $V(B_i) \setminus \partial B_i$. We then make a two-dimensional array $d_i[1, \ldots, |\partial B_i|][1, \ldots, |V(B_i)|]$ of size $O(r^{1/2}) \times O(r) = O(r^{3/2})$. Using Lemma 9 we compute the distances $d_G(u, v)$ for all $(u, v) \in (\partial B_i) \times V(B_i)$, and store the distance $d_G(u, v)$ in the entry $d_i[\phi_i(u)][\phi_i(v)]$. With this information, we can clearly access the distance $d_G(u, v)$ in $O(1)$ time for any query $(u, v) \in (\partial B_i) \times V(B_i)$. This sets part (e) of $\mathtt{DS}(G, r)$. We have spent $O(n \log n + r^{3/2})$ time and $O(n + r^{3/2})$ space per piece $B_i \in \mathbb{B}$.

Part (f) of $\mathtt{DS}(G, r)$ can be constructed using Theorem 7 in each piece $B_i \in \mathbb{B}$. This takes $O(r^{3/2})$ time and space per piece $B_i \in \mathbb{B}$.

Part (g) of $\mathtt{DS}(G, r)$ can be constructed using Theorem 4 for each hole $H_i^j = H(B_i, W^j)$ with respect to the face of H_i^j that contains the vertices $V(W^j)$. When we obtain a query $(u, v) \in (\partial B_i \cap V(H_i^j)) \times V(H_i^j)$, we just use the data structure of Theorem 4 to report $d_{H_i^j}(u, v)$. To bound the running time used to construct part (g) of $\mathtt{DS}(G, r)$, note that the holes $H_i^1, \ldots, H_i^{w_i}$ of piece B_i are pairwise edge-disjoint, and therefore applying Theorem 4 to the holes $H_i^1, \ldots, H_i^{w_i}$ takes $O(n \log n)$ time overall. This sets part (g) of $\mathtt{DS}(G, r)$. We have spent $O(n \log n)$ time and space per piece $B_i \in \mathbb{B}$.

We have finished the description of how to construct parts (a)–(g) of $\mathtt{DS}(G, r)$. Besides the $O(n \log n)$ time to construct the r-decomposition, we have used $O(n \log n + r^{3/2})$ time per piece $B_i \in \mathbb{B}$. Adding over all $O(n/r)$ pieces we see that the total time needed is

$$\sum_{B_i \in \mathbb{B}} O(n \log n + r^{3/2}) = O((n/r)(n \log n + r^{3/2})) = O((n^2/r) \log n + n r^{1/2}).$$

The same bound applies to the space used by the data structure. $\qquad\square$

Lemma 11 *For any query pair $(u, v) \in V(G)^2$, the data structure $\mathtt{DS}(G, r)$ can report in $O(r^{1/2} \log n)$ time the distance $d_G(u, v)$.*

Proof. Consider a query pair (u, v). Using the array $\mathtt{b}[\,]$ (part (b) of $\mathtt{DS}(G, r)$) we can identify a piece $B_{\mathtt{b}[u]} \in \mathbb{B}$ that contains vertex u. Using the entry $\mathtt{Locate}_{\mathtt{b}[u]}[v]$ (part (d) of

$\mathtt{DS}(G, r))$ we can decide if the vertex v is also in the piece $B_{\mathtt{b}[u]}$ or not. This gives rise to two cases.

Case $v \notin V(B_{\mathtt{b}[u]})$. In this case v is in some hole of $B_{\mathtt{b}[u]}$. Let $H_{\mathtt{b}[u]}^{j(v)}$ be the hole that contains v. We can identify $H_{\mathtt{b}[u]}^{j(v)}$ using the entry $\mathtt{Locate}_{\mathtt{b}[u]}[v]$ (part (d) of $\mathtt{DS}(G, r)$). We claim that

$$d_G(u, v) = \min\Big\{ d_G(u, v') + d_{H_{\mathtt{b}[u]}^{j(v)}}(v', v) \mid v' \in (\partial B_{\mathtt{b}[u]}) \cap V(H_{\mathtt{b}[u]}^{j(v)}) \Big\}.$$

Assuming the correctness of the claim, we can then compute $d_G(u, v)$ in $O(r^{1/2} \log n)$ time: we go through the linked list $L_{\mathtt{b}[u]}^{j(v)}$ (part (c) of $\mathtt{DS}(G, r)$) containing the vertices $(\partial B_{\mathtt{b}[u]}) \cap V(H_{\mathtt{b}[u]}^{j(v)})$, and for each vertex v' in $L_{\mathtt{b}[u]}^{j(v)}$, we obtain the value $d_G(u, v')$ (resp. $d_{H_{\mathtt{b}[u]}^{j(v)}}(v', v)$) in $O(1)$ (resp. $O(\log n)$) time using part (e) (resp. (g)) of $\mathtt{DS}(G, r)$.

To see the correctness of the claim, we apply an idea similar to the ones in previous lemmas. Clearly, we have

$$d_G(u, v) \leq \min\Big\{ d_G(u, v') + d_{H_{\mathtt{b}[u]}^{j(v)}}(v', v) \mid v' \in (\partial B_{\mathtt{b}[u]}) \cap V(H_{\mathtt{b}[u]}^{j(v)}) \Big\}.$$

For the other inequality, consider a shortest path $\Delta_G(u, v)$ in G between u and v, and let v'' be the last vertex of $\Delta_G(u, v)$ in $\partial B_{\mathtt{b}[u]}$. The portion of $\Delta_G(u, v)$ between v'' and v has to be contained in $H_{\mathtt{b}[u]}^{j(v)}$, and v'' has to be a vertex of $(\partial B_{\mathtt{b}[u]}) \cap V(H_{\mathtt{b}[u]}^{j(v)})$. Therefore

$$d_G(u, v) = d_G(u, v'') + d_G(v'', v) = d_G(u, v'') + d_{H_{\mathtt{b}[u]}^{j(v)}}(v'', v)$$

$$\geq \min\{ d_G(u, v') + d_{H_{\mathtt{b}[u]}^{j(v)}}(v', v) \mid v' \in (\partial B_{\mathtt{b}[u]}) \cap V(H_{\mathtt{b}[u]}^{j(v)}) \}.$$

This finishes the proof of the claim and closes the case $v \notin V(B_{\mathtt{b}[u]})$.

Case $v \in V(B_{\mathtt{b}[u]})$. We claim that

$$d_G(u, v) = \min \left(\{ d_{B_i}(u, v) \} \cup \{ d_G(u, v') + d_G(v', v) \mid v' \in \partial B_i \} \right).$$

Assuming the correctness of the claim, we can then compute the value $d_G(u, v)$ in $O(r^{1/2})$ time: the value $d_{B_i}(u, v)$ can be obtained using part (f) of $\mathtt{DS}(G, r)$, and the value $\min \{ d_G(u, v') + d_G(v', v) \mid v' \in \partial B_i \}$ can be obtained going through the linked list containing the vertices of $\partial B_{\mathtt{b}[u]}$ (part (a) of $\mathtt{DS}(G, r)$), and for each vertex $v' \in \partial B_{\mathtt{b}[u]}$ obtaining the values $d_G(u, v')$, $d_G(v', v)$ in $O(1)$ time using part (e) of $\mathtt{DS}(G, r)$.

To see the correctness of the claim, we apply the same idea as above. Clearly, we have

$$d_G(u, v) \leq \min \left(\{ d_{B_i}(u, v) \} \cup \{ d_G(u, v') + d_G(v', v) \mid v' \in \partial B_i \} \right).$$

For the other inequality, consider a shortest path $\Delta_G(u, v)$ in G between u and v. If $\Delta_G(u, v)$ is contained in B_i, then

$$d_G(u, v) = d_{B_i}(u, v) \geq \min \left(\{ d_{B_i}(u, v) \} \cup \{ d_G(u, v') + d_G(v', v) \mid v' \in \partial B_i \} \right).$$

If $\Delta_G(u, v)$ is not contained in B_i, let v'' be any vertex of $\Delta_G(u, v) \cap \partial B_i$. We then have

$$d_G(u, v) = d_G(u, v'') + d_G(v'', v)$$

$$\geq \min \left(\{ d_{B_i}(u, v) \} \cup \{ d_G(u, v') + d_G(v', v) \mid v' \in \partial B_i \} \right).$$

This finishes the proof of the claim and closes the case $v \in V(B_{\mathtt{b}[u]})$.

$\square$

We still can make the choice for the best value r in the data structure $\text{DS}(G, r)$. Constructing $\text{DS}(G, r)$ takes $O((n^2/r) \log n + nr^{1/2})$ time and space. Observe that when $r \leq n^{2/3} \log^{2/3} n$, the first term of the sum dominates, while when $r \geq n^{2/3} \log^{2/3} n$, the second one dominates. In any case, we cannot expect to bound the time and space complexity to construct $\text{DS}(G, r)$ below $O(n^{4/3} \log^{1/3} n)$. By choosing r, we get a family of data structures with a trade-off between the time for its construction and their time to answer queries.

Theorem 12 *Let G be a planar graph with n vertices. For any value S in the interval $[n^{4/3} \log^{1/3} n, n^2]$, we can construct in $O(S)$ time a data structure of size $O(S)$ that answers distance queries in $O\left((n/\sqrt{S}) \log^{3/2} n\right)$ time per query.*

Proof. Construct $\text{DS}(G, r)$ for $r = (n^2/S) \log n$. According to Lemma 10, the construction of $\text{DS}(G, r)$ takes

$$O\left(\frac{n^2 \log n}{(n^2/S) \log n} + n\sqrt{(n^2/S) \log n}\right) = O\left(S + (n^2/\sqrt{S}) \log^{1/2} n\right) = O(S),$$

where in the last step we used $S \geq n^{4/3} \log^{1/3} n$. For answering a query, Lemma 11 shows that it takes $O(\sqrt{r} \log n) = O((n/\sqrt{S})\sqrt{\log n} \log n) = O((n/\sqrt{S}) \log^{3/2} n)$ time per query. $\square$

5 Many distances in planar graphs

Using Theorem 12 we can prove the following result.

Theorem 13 *Let G be a planar graph of size n. The* online *k-many distances problem in G can be solved in $O(k^{2/3} n^{2/3} \log n + n^{4/3} \log^{1/3} n)$ time, even when we do not know the value k beforehand.*

Proof. Set $k_0 = n/\log n$, and $k_i = k_0 2^i$ for any positive integer i. We use $r_i = n^{4/3} k_i^{-2/3}$ for any integer i.

For the first k_0 pairs that we have to answer, we use the data structure $\text{DS}(G, r_0)$. It takes

$$
\begin{aligned}
O\left(\frac{n^2 \log n}{r_0} + nr_0^{1/2}\right) &= O\left(\frac{n^2 \log n}{n^{4/3} k_0^{-2/3}} + n\sqrt{n^{4/3} k_0^{-2/3}}\right) \\
&= O\left(n^{2/3} \log n \, (n/\log n)^{2/3} + n^{5/3} (n/\log n)^{-1/3}\right) \\
&= O\left(n^{4/3} \log^{1/3} n\right)
\end{aligned}
$$

time to construct $\text{DS}(G, r_0)$, and each distance query can be answered in

$$O(\sqrt{r_0} \log n) = O(\sqrt{n^{4/3} k_0^{-2/3}} \log n) = O(n^{2/3} (n/\log n)^{-1/3} \log n) = O(n^{1/3} \log^{4/3} n)$$

time. Therefore, the distance for the first k_0 pairs can be computed in $O(n^{4/3} \log^{1/3} n)$ time.

Next, the rule we apply is as follows. Each time that the number of pairs we have received reaches a value k_i, we construct the data structure $\text{DS}(G, r_i)$ and answer the distance queries for the next pairs using it. That is, the pairs that we receive between the k_i-th and the $(2k_i - 1)$-th pair are answered using $\text{DS}(G, r_i)$.

In total, if $k \geq n/\log n$, we will then construct the data structures $\mathtt{DS}(G, r_i)$ for $i = 0, 1, \ldots, \lfloor \log_2(k/k_0) \rfloor$. This takes

$$\sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} \text{Time to construct } \mathtt{DS}(G, r_i) = \sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} O\left(\frac{n^2 \log n}{r_i}\right)$$

$$= O\left(\sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} \frac{n^2 \log n}{n^{4/3} k_i^{-2/3}}\right)$$

$$= O\left(\sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} n^{2/3} k_i^{2/3} \log n\right)$$

$$= O\left(n^{2/3} \log n \sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} (k_0 2^i)^{2/3}\right)$$

$$= O\left(n^{2/3} k_0^{2/3} \log n \sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} 2^{2i/3}\right)$$

$$= O\left(n^{2/3} k_0^{2/3} (k/k_0)^{2/3} \log n\right)$$

$$= O\left(n^{2/3} k^{2/3} \log n\right)$$

time.

As for the time needed for computing the distances, observe that the k_i pairs between the k_i-th and the $(k_{i+1} - 1)$-th are answered using $\mathtt{DS}(G, r_i)$. Each distance in $\mathtt{DS}(G, r_i)$ takes

$$O(\sqrt{r_i} \log n) = O(\sqrt{n^{4/3} k_i^{-2/3}} \log n) = O(n^{2/3} k_i^{-1/3} \log n)$$

time, and therefore computing the k_i distances in $\mathtt{DS}(G, r_i)$ takes $O(n^{2/3} k_i^{2/3} \log n)$ time.

The total time we spend computing all the distances is

$$\sum_{i=0}^{\lfloor \log_2(k/k_0) \rfloor} O\left(n^{2/3} k_i^{2/3} \log n\right)$$

which we have just seen above that is $O(n^{2/3} k^{2/3} \log n)$. $\qquad\square$

One weak point in this approach is that we are not indeed using that the k pairs are known beforehand. That is, we are not taking profit that the problem k-many distances, as we have defined it, is an off-line problem. Using this feature, we only know how to improve the space bound.

Theorem 14 *Let G be a planar graph of size n. The problem k-many distances in G can be solved in $O(k^{2/3} n^{2/3} \log n + n^{4/3} \log^{1/3} n)$ time and $O(n + k)$ space.*

Proof. Let $P = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be the pairs of vertices for which we want to compute the distance. We assume that $k \geq n/\log n$, as otherwise we can just add extra pairs of vertices and the time bound to be proved remains the same while the space bound becomes stronger. Consider a parameter $r = n^{4/3} k^{-2/3}$. We interleave the construction of $\mathtt{DS}(G, r)$ and the queries to be answered, as follows.

We use Theorem 3 to construct an r-decomposition with pieces $\mathbb{B} = \{B_1, \ldots B_p\}$, where $p = \Theta(n/r)$. We keep using w_i to denote the number of boundary walks of piece $B_i \in \mathbb{B}$.

14

It holds that $\sum_i w_i = O(n/r)$. We construct parts (a) and (b) of $\mathtt{DS}(G,r)$, which require $O(n \log n)$ time and $O(n)$ space; see the first paragraph in the proof of Lemma 10.

We split the pairs in P into groups P_i, $i = 1 \ldots p$, such that the following holds: if a pair (s,t) is in P_i, then s is a vertex in the piece B_i. This split can be done in $O(k)$ time by assigning the pair $(s,t) \in P$ to the group $P_{b[s]}$. Let $k_i = |P_i|$. We further split each group P_i into subgroups $P_i^1, P_i^2, \ldots, P_i^{\lceil k_i r^{1/2}/n \rceil}$ of $n/r^{1/2}$ pairs each, except possibly the last subgroup.

Consider any fixed subgroup P_i^a. To simplify notation, let us take $Q = P_i^a$. We next discuss how to compute $d_G(s_z, t_z)$ for all pairs (s_z, t_z) in the subgroup Q. Let T denote the vertices $\{ t_z \mid (s_z, t_z) \in Q \}$.

We construct parts (c)–(f) for the piece B_i. This can be done in $O(n \log n + r^{3/2})$ time and $O(n)$ space as it is discussed in the proof of Lemma 10. We also attach to each vertex v of ∂B_i a distinct label $\phi_\partial(v) \in \{1, \ldots, |V(B_i)|\}$ and to each vertex t_z of T a distinct label $\phi_T(t) \in \{1, \ldots, |T|\}$. That is, ϕ_∂ is a bijection between ∂B_i and $\{1, \ldots, |\partial B_i|\}$ and ϕ_T is a bijection between T and $\{1, \ldots, |T|\}$. This can be done in $O(n)$ time by traversing B_i and G.

We make a table $\delta[1, \ldots, |\partial B_i|][1, \ldots, |T|]$ of size $O(r^{1/2} \cdot n/r^{1/2}) = O(n)$ whose entries are settled to $+\infty$ at the beginning. For each pair $(v, t_z) \in \partial B_i \times T$ we want to achieve that $\delta[\phi_\partial(v)][\phi_T(t_z)]$ is $d_G(v, t_z)$ if $t_z \in V(B_i)$ and $d_{H_i^j}(v, t_z)$ if t_z is in $V(H_i^j) \setminus V(B_i)$. This is achieved as follows. We split T into subgroups $T^0, T^1, \ldots, T^{w_i}$, where T^0 contains the vertices t_z in B_i, and T^j contains the vertices t_z in $V(H_i^j) \setminus V(B_i)$. This split can be done in $O(|T|) = O(n/r^{1/2})$ time by using $\mathtt{Locate}_i[t_z]$ for each t_z of T. The distances $d_G(v, t_z)$ for all pairs $(v, t_z) \in \partial B_i \times T^0$ can be obtained and stored into $\delta[\phi_\partial(v)][\phi_T(t_z)]$ in $O(|\partial B_i| \cdot |T^0|) = O(r^{1/2}|T^0|)$ time using part (e) of $\mathtt{DS}(G,r)$. The distances $d_G(v, t_z)$ for all pairs $(v, t_z) \in \partial B_i \times T^j$ can be obtained and stored into $\delta[\phi_\partial(v)][\phi_T(t_z)]$ using the second part of Theorem 4 in

$$O((|V(H_i^j)| + |\partial B_i| \cdot |T^j|) \log n) = O((|V(H_i^j)| + r^{1/2}|T^j|) \log n)$$

time and

$$O(|V(H_i^j)| + |\partial B_i| \cdot |T^j|) = O(n + r^{1/2} \cdot n/r^{1/2}) = O(n)$$

space. Since we can reuse space for each hole H_i^j, we conclude that the table δ can be filled in using $O(n)$ space and

$$O\left(r^{1/2}|T^0| + \sum_{j=1}^{w_i} (|V(H_i^j)| + r^{1/2}|T^j|) \log n \right)$$

time. This time can be bounded by

$$
\begin{aligned}
O\left(\sum_{j=1}^{w_i} |V(H_i^j)| \log n + \sum_{j=0}^{w_i} r^{1/2}\, |T^j| \log n \right) &= O(n \log n + r^{1/2}|T| \log n) \\
&= O(n \log n + r^{1/2}\, (n/r^{1/2}) \log n) \\
&= O(n \log n).
\end{aligned}
$$

time.

When the table δ is available, we can compute the distances $d_G(s_z, t_z)$ for all pairs $(s_z, t_z) \in Q$ using the approach of Lemma 11. Any distance that is needed can be recovered in $O(1)$ time from the table δ, and hence we spend $O(r^{1/2})$ time and $O(1)$ additional space per

distance. We conclude that we can compute $d_G(s_z, t_z)$ for all pairs $(s_z, t_z) \in Q$ in $O(n \log n)$ time and $O(n)$ space.

Repeating the procedure for each group P_i^a, where $a = 1, \ldots, \lceil k_i r^{1/2}/n \rceil$ and $i = 1 \ldots p$, we can compute the distance for all pairs P using

$$
\begin{aligned}
\sum_i \sum_{a=1}^{\lceil k_i r^{1/2}/n \rceil} O(n \log n) &= \sum_i O(k_i r^{1/2}/n) \cdot O(n \log n) \\
&= \sum_i O(k_i r^{1/2} \log n) \\
&= O(k r^{1/2} \log n) \\
&= O(n^{2/3} k^{2/3} \log n)
\end{aligned}
$$

time. Since for each group P_i^a we can reuse the working space of size $O(n)$, we only need $O(n + k)$ space in total. The result follows. $\qquad\square$

6   Stretch factor of planar geometric graphs

An Euclidean graph is a graph whose vertices are embedded in some Euclidean space $\mathbb{R}^d$ and such that the length of each edge is the Euclidean distance between its vertices. Given an Euclidean graph G, one of its relevant parameters is its stretch factor, defined as

$$
t_G = \max_{u,v \in V(G)} \left\{ \frac{d_G(u,v)}{|uv|} \right\},
$$

where $|\cdot|$ denotes the Euclidean distance. This parameter measures how well the distances in the graph resemble the Euclidean distances, and is the key parameter for the construction of geometric spanners [7]. Narasimhan and Smid have shown the following result.

Theorem 15 [19] Given an Euclidean graph G in $\mathbb{R}^d$, and a parameter $3 \geq \varepsilon > 0$, computing a value t such that $t \leq t_G \leq (1 + \varepsilon)t$ takes $O(n \log n)$ time plus that the time to answer $O(\varepsilon^{-d}n)$-many distances in G.

When the graph G is planar, although not necessarily its geometric drawing, then we can use Theorem 14 to conclude the following result.

Theorem 16 Let G be an Euclidean graph with n vertices in $\mathbb{R}^d$, let $3 \geq \varepsilon > 0$ be a parameter, and assume that G is planar. We can compute in $O(\varepsilon^{-2d/3} n^{4/3} \log n)$ time a value t such that $t \leq t_G \leq (1 + \varepsilon)t$.

Observe that if we fix the parameter ε to a constant value, we obtain a running time of $O(n^{4/3} \log n)$. This represents a considerable improvement over the previous running time of $O(n^{3/2})$ [19]. On the other hand, if we restrict ourselves to families of graphs whose dilations t_G are bounded by a constant, and we also consider ε to be constant, Gudmundsson et al. [11] have shown how to compute in $O(n \log n)$ time a value t such that $t \leq t_G \leq (1 + \varepsilon)t$. This latter result is applicable to arbitrary graphs.

7   Discussion

We have presented data structures and algorithms for computing distances in planar graphs with non-negative edge-lengths. The algorithms and data structures can easily be extended

to directed planar graphs. It also extends to directed graphs with negative and positive weights, assuming that there are no cycles of negative length. For this, we just convert the problem to a problem involving positive weights by computing all the distances from an arbitrary source in $O(n \log^2 n)$ time and then defining a feasible price function; see [13] or [8]).

The main open problem concerns finding the complexity of the k-many distances problem. In particular, can it be solved in roughly $O(n+k)$ time? Near-linear time answers are known only when $k = O(\sqrt{n})$ or $k = \Theta(n^2)$. A first approach towards this problem may be considering the problem of computing the pairwise distances between a set of k vertices in a planar graph. Currently, the best result when $k = \Theta(\sqrt{n})$ is achieved by reducing it to a $\Theta(k^2)$-many distances problem.

References

[1] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *ESA '96*, volume 1136 of *LNCS*, pages 514–528. Springer, 1996.

[2] S. Cabello. Many distances in planar graphs. In *SODA '06: Proc. 17th Symp. Discrete algorithms*, pages 1213–1220. ACM Press, 2006.

[3] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. In *SODA '07: Proc. 18th Symp. Discrete Algorithms*, pages 89–97, 2007.

[4] T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. Preliminary version in WADS '05.

[5] D. Z. Chen and J. Xu. Shortest path queries in planar graphs. In *STOC '00: Proceedings of the 32nd annual ACM symposium on Theory of computing*, pages 469–478, 2000.

[6] H. N. Djidjev. Efficient algorithms for shortest path problems on planar digraphs. In F. d'Amore, P. G. Franciosa, and A. Marchetti-Spaccamela, editors, *WG'96*, volume 1197 of *LNCS*, pages 151–165. Springer, 1997.

[7] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. Elsevier, 2000.

[8] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

[9] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[10] G. N. Frederickson. Planar graph decomposition and all pairs shortest paths. *J. ACM*, 38(1):162–204, 1991.

[11] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric spanners. *ACM Trans. Algorithms*, 4(1):1–34, 2008.

[12] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[13] P. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n \log^2 n)$-time algorithm. *ACM Trans. Algorithms*, accepted. Preliminary version in SODA'09.

[14] P. N. Klein. Multiple-source shortest paths in planar graphs. In *SODA '05: Proc. 16th Symp. Discrete algorithms*, pages 146–155, 2005.

[15] L. Kowalik and M. Kurowski. Short path queries in planar graphs in constant time. In *STOC '03: Proc. 35th Symp. Theory of computing*, pages 143–148, 2003.

[16] M. Kutz. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In *SOCG '06: Proc. 22nd Symp. Comput. Geom.*, pages 430–438, 2006.

[17] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[18] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32:265–279, 1986.

[19] G. Narasimhan and M. Smid. Approximating the stretch factor of euclidean graphs. *SIAM Journal on Computing*, 30:978–989, 2000.

[20] S. Tazari and M. Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Applied Mathematics*, in press.

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

[22] U. Zwick. Exact and Approximate Distances in Graphs - A Survey. In *ESA 2001*, volume 2161 of *LNCS*, pages 33–48. Springer, 2001.