Near-Optimal Distance Emulator for Planar Graphs

Hsien-Chih Chang[1]
University of Illinois at Urbana-Champaign, USA
hchang17@illinois.edu

Paweł Gawrychowski
University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Shay Mozes[2]
IDC Herzliya, Israel
smozes@idc.ac.il

Oren Weimann[3]
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

Given a graph G and a set of terminals T, a *distance emulator* of G is another graph H (not necessarily a subgraph of G) containing T, such that all the pairwise distances in G between vertices of T are preserved in H. An important open question is to find the smallest possible distance emulator.

We prove that, given any subset of k terminals in an n-vertex undirected unweighted planar graph, we can construct in $\tilde{O}(n)$ time a distance emulator of size $\tilde{O}(\min(k^2, \sqrt{k \cdot n}))$. This is optimal up to logarithmic factors. The existence of such distance emulator provides a straightforward framework to solve distance-related problems on planar graphs: Replace the input graph with the distance emulator, and apply whatever algorithm available to the resulting emulator. In particular, our result implies that, on any unweighted undirected planar graph, one can compute all-pairs shortest path distances among k terminals in $\tilde{O}(n)$ time when $k = O(n^{1/3})$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases planar graphs, shortest paths, metric compression, distance preservers, distance emulators, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.16

1 Introduction

The planar graph metric is one of the most well-studied metrics in graph algorithms, optimizations, and computer science in general. The planar graph *metric compression* problem is to compactly represent the distances among a subset T of k vertices, called ***terminals***, in an n-vertex planar graph G. Without compression (and when G is unweighted) these distances can be naïvely represented with $O(\min(k^2 \log n, n))$ bits by either explicitly storing the $k \times k$ distances or alternatively by storing the entire graph (naïvely, this takes $O(n \log n)$ bits, but can be done with $O(n)$ bits [13, 24, 57, 62]).

[1] Supported partially by NSF grant CCF-1408763.
[2] Supported partially by ISF grants 794/13 and 592/17
[3] Supported partially by ISF grants 794/13 and 592/17

A natural way to compress G is to replace it by another graph H that contains T as vertices, and the distances between vertices in T are preserved in H. In other words, $d_G(x,y) = d_H(x,y)$ holds for every pair of vertices x and y in T. Such graph H is called a ***distance emulator*** of G with respect to T (or a ***distance preserver*** in the case where H is required to be a subgraph of G). Distance emulators are appealing algorithmically, since we can readily feed them into our usual graph algorithms. They have been studied as compact representations of graphs [8,14,15,16,25,26], and used as fundamental building blocks in algorithms and data structures for distance-related problems [1,2,3,16,28,31,43]. Similar concepts like additive and multiplicative *spanners* [11,12,28,41,59,65] and *distance labelings* [9,10,35,36,40,46,55,64] are popular topics with abundant results.

In planar graphs, an extensive study has been done for the case where the emulator H is required to be a minor of G [11,21,23,30,32,38,39,41,53]. Restricting H to be a minor of G has proven useful when H is only required to *approximate* the distances between vertices in T, say, up to a $(1+\varepsilon)$ multiplicative error. For exact distances however, Krauthgamer, Nguyen, and Zondiner [52] have shown a lower bound of $\Omega(k^2)$ on the size of H, even when G is an unweighted grid graph and H is a (possibly weighted) minor emulator.

In general, an emulator H does not have to be a subgraph or a minor of G. In fact, H can be non-planar even when G is planar. Even in this setting, for *weighted* planar graphs, we cannot beat the naïve bound because one can encode an arbitrary $k \times k$ binary matrix using subset distances among $2k$ vertices in a weighted planar graph [4,35]. Since we cannot compress an arbitrary binary $k \times k$ matrix into less than k^2 bits, we again have an $\Omega(k^2)$ lower bound.

What about *unweighted* planar graphs? Various distance-related problems in unweighted graphs enjoy algorithms and techniques [12,22,28,29,51,59,63,65,66,67] which outperform their weighted counterparts. Indeed, for the metric compression problem in unweighted planar graphs, Abboud, Gawrychowski, Mozes, and Weimann [4] have very recently shown that we can in fact beat the $O(k^2)$ bound. They showed that the distances between any k terminals can be encoded using $\tilde{O}(\min(k^2, \sqrt{k \cdot n}))$ bits[4]. The encoding of Abboud *et al.* is optimal up to logarithmic factors. However, it is not an emulator! Goranci, Henzinger, and Peng [38] raised the question of whether such encoding can be achieved by an emulator. We answer this question in the affirmative.

Our results. We show that the encoding of Abboud *et al.* can be turned into an emulator, with no asymptotic overhead. Namely, we prove that any unweighted planar graph has a near-optimal size distance emulator.

▶ **Theorem 1.** *Let G be an n-vertex undirected unweighted planar graph, and let T be the set of k terminals. A directed weighted graph H of size $O(\min(k^2, \sqrt{k \cdot n}\log^3 n))$ can be constructed in $O(n \log^4 n)$ time as a distance emulator of G with respect to T; all edge weights of H are non-negative integers bounded by n.*

Our theorem provides a practical framework for solving distance-related problems on planar graphs: Replace the input graph G (or proper subgraphs of the right choice) with the corresponding distance emulator H, and invoke whatever algorithm available on H rather than on G. One concrete example of this is the computation of all-pairs shortest paths among a subset T of k vertices. The algorithm by Cabello [19] can compute these $\Theta(k^2)$ distances in $\tilde{O}(n^{4/3} + n^{2/3}k^{4/3})$ time. Mozes and Sommer [56] improved the running time to $\tilde{O}(n^{2/3}k^{4/3})$ when $k = \Omega(n^{1/4})$. Our emulator immediately implies a further improvement of the running time to $\tilde{O}(n + n^{1/2}k^{3/2})$ using

[4] The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in n.

an extremely simple algorithm: Replace G by the emulator H and run Dijkstra's single-source shortest-paths algorithm on H from every vertex of T. This improves on previous results by a polynomial factor for essentially the entire range of k.

▶ **Corollary 2.** *Let G be an undirected unweighted planar graph, and let T be a subset of k terminals in G. One can compute the shortest path distances between all pairs of vertices in T in $O(n\log^4 n + n^{1/2}k^{3/2}\log^3 n \log\log n)$ time.*

Techniques. The main novel idea is a graphical encoding of unit-Monge matrices. Our emulator is obtained by plugging this graphical encoding into the encoding scheme of Abboud *et al.* [4]. Their encoding consists of two parts. First, they explicitly store a set of distances between some pairs of vertices of G (not necessarily of T). It is trivial to turn this part into an emulator by adding a single weighted edge between every such pair. Second, they implicitly store all-pairs distances in a set containing pairs of the form (X,Y) where X is a *prefix* of a cyclic walk around a single face and Y is a *subset* of vertices of a cyclic walk around a (possibly different) single face. For each such pair (X,Y), the X-to-Y distances are efficiently encoded using only $O((|X|+|Y|)\cdot\log n)$ bits (rather than the naïve $O(|X|\cdot|Y|\cdot\log n)$ bits). This encoding follows from the fact that the $X\times Y$ distance matrix M is a *triangular unit-Monge* matrix.

A matrix is ***Monge*** if for any i,j we have that

$$M[i+1,j]-M[i,j] \;\leq\; M[i+1,j+1]-M[i,j+1].$$

A Monge matrix is ***unit-Monge*** if for any i,j we also have that

$$M[i+1,j]-M[i,j] \;\in\; \{-1,0,1\}.$$

A (unit-)Monge matrix is ***triangular*** if the above conditions are only required to hold for $i\neq j$. In other words, when all four entries $M[i,j], M[i+1,j], M[i,j+1], M[i+1,j+1]$ belong to the upper triangle of M or to the lower triangle of M.

The Monge property has been heavily utilized in numerous algorithms for distance related problems in planar graphs [17, 18, 20, 33, 44, 49, 54, 56] in computational geometry [5, 6, 7, 37, 47, 48], and in pattern matching [27, 42, 58, 61]. However, prior to the work of Abboud *et al.* [4], the stronger *unit-Monge* property has only been exploited in pattern matching [60, 61].

The encoding of the $X\times Y$ unit-Monge matrix M is immediate: For any i, the sequence of differences $M[i+1,j]-M[i,j]$ is nondecreasing and contains only values from $\{-1,0,1\}$, so they can be encoded by storing the positions of the first 0 and the first 1. Storing these positions for every i takes $O(|X|\cdot\log n)$ bits. To encode M, we additionally store $M[0,j]$ for every j using $O(|Y|\cdot\log n)$ bits. This encoding, however, is clearly not an emulator. Our main technical contribution is in showing that it can be turned into an emulator with no asymptotic overhead. Namely, in Section 2 we prove the following lemma.

▶ **Lemma 3.** *Given an $n\times n$ unit-Monge or triangular unit-Monge matrix M, one can construct in $O(n\log n)$ time a directed edge-weighted graph (emulator) H with $O(n\log n)$ vertices and edges such that the distance in H between vertex i and vertex j is exactly $M[i,j]$.*

In Section 3 we describe how to plug the emulator of Lemma 3 into the encoding scheme of Abboud *et al.* [4]. In their paper, the size of the encoding was the only concern and the construction time was not analyzed. In our case, however, the construction time is crucial for the application in Corollary 2. We achieve an $O(n\log^4 n)$ time construction by making a small modification to their encoding, based on the technique of heavy path decomposition.

Another difference is that in their construction once the encoding is computed, single-source shortest-paths can be computed on the encoding itself using an algorithm by Fakcharoenphol and Rao [33]. More precisely (see [4, Section 5]), using a compressed representation of unit-Monge matrices, the total size of their encoding is $s = O(\sqrt{k \cdot n} \log^2 n)$ words, running the Fakcharoenphol and Rao algorithm requires accessing $O(s \log^2 s)$ elements of the unit-Monge matrices, and accessing each such element from the compressed representation requires $O(\log n / \log\log n)$ time (via a range dominance query, see Section 2)[5]. Overall, using our $O(n \log^4 n)$ construction together with the above single-source shortest-paths computations from each terminal would result in a time bound of $O(n \log^4 n + n^{1/2} k^{3/2} \log^5 n / \log\log n)$ in Corollary 2. Our approach on the other hand, runs Dijkstra's classical algorithm on the emulator H, leading to a faster (by a $(\log n / \log\log n)^2$ factor) time bound for Corollary 2. More precisely, our emulator H is of size $O(\sqrt{k \cdot n} \log^3 n)$, the edge-weights in H are encoded explicitly (i.e. there is no need for a range dominance query), and the edge-weights are integers in $[1 \,..\, n]$ so Dijkstra's algorithm can use an $O(\log\log n)$ time heap (using such fast heap is not possible for Fakcharoenphol and Rao's algorithm). Running Dijkstra therefore takes $O(\sqrt{k \cdot n} \log^3 n \log\log n)$ time leading to the bound in Corollary 2.

2 Distance Emulators for Unit-Monge Matrices

2.1 The bipartite case

We next describe an $O(n \log n)$ space and time construction of a distance emulator for *square* $n \times n$ unit-Monge matrices. The construction can be trivially extended to *rectangular* $n \times m$ unit-Monge matrices (in $O(\max(n,m)\log(\max(n,m)))$ time and space) by duplicating the last row or column until the matrix becomes square.

We begin by establishing a relation between unit-Monge matrices and right substochastic binary matrices. A binary matrix P is ***right substochastic*** if every row of P contains at most one nonzero entry. The following lemma can be established similarly to Abboud *et al.* [4, Lemma 6.1].[6]

▶ **Lemma 4** (Abboud *et al.* [4, Lemma 6.1])**.** *For any $n \times m$ unit-Monge matrix M there is a right substochastic $2(n-1) \times (m-1)$ binary matrix P such that for all x and y,*

$$M[x,y] \;=\; U[x] + V[y] + \sum_{\substack{i \geq 2x-1 \\ j \geq y}} P[i,j]$$

where U is a vector of length n and V is a vector of length m.

Proof. We first define an $(n-1) \times (m-1)$ matrix P' as follows:

$$P'[i,j] = M[i+1,j+1] + M[i,j] - M[i,j+1] - M[i+1,j].$$

By Monge, clearly $P'[i,j] \geq 0$, and by unit $P'[i,j] \leq 2$. In fact, unit also implies that the sum in every row of P' is at most 2. To see this, consider $P'[i,1] + P'[i,2] + \ldots + P'[i,n]$. After telescoping, this is $P'[i,1] - P[i+1,1] + P[i+1,m] - P[i,m]$, so by unit at most 2 as claimed. Then, consider $\sum_{i' \geq i, j' \geq j} P'[i',j']$. After substituting the definition of P' and telescoping, this

[5] The $O(\log n / \log\log n)$ factor was overlooked in [4]. It can be obtained by representing the unit-Monge matrix with a range dominance data structure, such as the one in [45].

[6] In Abboud *et al.* [4, Section 6] a unit-Monge matrix is defined to be one where every two adjacent elements differ by at most 1, whereas here we only assume this for elements that are adjacent vertically (that is, in the same column and between adjacent rows).

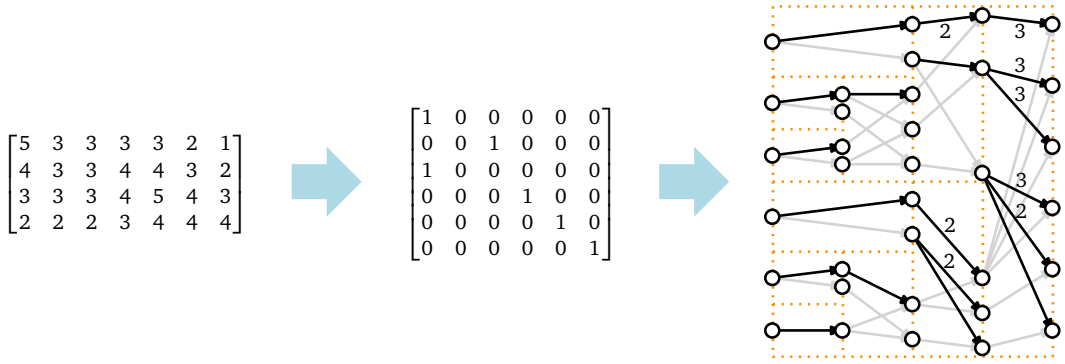

Figure 1 Unit-Monge distance matrix M, its corresponding row substochastic matrix P, and the emulator H for the matrix P. In this example the two vectors given by Lemma 4 are $U = [-3\ -2\ -1\ 0]$ and $V = [2\ 2\ 2\ 3\ 4\ 4\ 4]$. The leftmost vertices in H are row vertices $r[x]$, and the rightmost vertices in H are column vertices $c[y]$. All gray directed edges without labels have weight 0, and all black directed edges without labels have weight 1. For instance, the distance from $r[2]$ to $c[1]$ is equal to the number of 1s in P dominated by the $(2,1)$-entry, which is 5.

becomes $M[i,j]+M[n,m]-M[i,m]-M[n,j]$. Hence, if we define $U[i]=M[i,m]-M[n,m]$ and $V[j]=M[n,j]$ it holds that $M[i,j]=U[i]+V[j]+\sum_{i'\geq i,j'\geq j}P'[i',j']$. Finally, we create a $2(n-1)\times(n-1)$ matrix P, where every 2×1 block corresponds to a single $P'[i,j]$, that is, the sum of values in the block is equal to $P'[i,j]$. It is always possible to define P so that it is a permutation matrix. To see this, consider a row of P'. The values there sum up to at most 2, say $P'[i,j]=P'[i,j']=1$ for some $j<j'$. Then, $P'[i,j]$ should correspond to a 1 in the first row of its block and $P'[i,j']$ to a 1 in the second row of its block. If $j=j'$ then in the corresponding block we create two 1s, one per row. ◀

Our goal is to construct a small emulator (in terms of number of vertices and edges) for M. By Lemma 4, it suffices to construct a graph H satisfying

$$d_H[r[x],c[y]] = \sum_{\substack{i\geq x\\ j\geq y}} P[i,j],$$

where x and y range over the rows and columns of P, $r[x]$ is a vertex corresponding to row x of P, $c[y]$ is a vertex corresponding to column y of P, and the $r[x]$-to-$c[y]$ distance in H equals the number of 1s in P dominated by entry $P[x,y]$ (a 2-dimensional range dominance counting). We assume that P is given as input, which is represented as a vector specifying, for each row i of P, the index of the (at most) single one entry in that row. We refer to such a graph H as an ***emulator*** for the right substochastic matrix P. To convert an emulator for P into an emulator for M, add a new vertex $r_0[x]$ for each $1\leq x<n$, and connect it with an edge of weight $U[x]$ to $r[2x-1]$. Similarly, for each $1\leq y<m$, connect $c[y]$ to a new vertex $c_0[y]$ with an edge of weight $V[y]$. In addition, add a new vertex $c_0[m]$ and connect each $r_0[x]$ to $c_0[m]$ by an edge of weight $U[x]+V[m]$; and add a new vertex $r_0[n]$ and connect $r_0[n]$ to each $c_0[y]$ by an edge of weight $U[n]+V[y]$. By Lemma 4, the $r_0[x]$-to-$c_0[y]$ distance equals $M[x,y]$. We therefore focus for the remainder of this section on constructing an emulator for the right substochastic matrix P.

Recursive construction. We now describe how to construct an emulator for an $n\times\alpha n$ right substochastic matrix P recursively for arbitrary constant $\alpha\geq 0$.

When P has only a single row, we create a row vertex r for the single row and a column vertex $c[y]$ for each column y. Connect r to $c[y]$ with a directed edge of weight $\sum_{j\geq y}P[1,j]$.

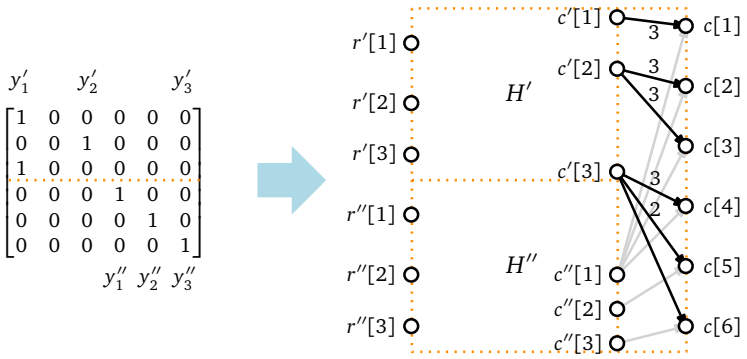

Figure 2 The top-level construction of the emulator. As an example, the edge from $c'[2]$ to $c[2]$ exists because $y'_{\rightarrow} = 2$ when $y = 2$, and the weight on the edge is equal to $\sum_{i>3, j\geq 2} P[i,j] = 3$.

Otherwise, assume P has more than one single row. Divide P into the top $\lfloor n/2 \rfloor \times \alpha n$ submatrix $\boldsymbol{P}_{\uparrow}$ and the bottom $\lceil n/2 \rceil \times \alpha n$ submatrix $\boldsymbol{P}_{\downarrow}$. Since $P_{\uparrow}$ is right substochastic, $P_{\uparrow}$ has at most $\lfloor n/2 \rfloor$ columns that are not entirely zero; similarly $P_{\downarrow}$ has at most $\lceil n/2 \rceil$ non-zero columns. Let $\boldsymbol{P'}$ and $\boldsymbol{P''}$ be the submatrices of $P_{\uparrow}$ and $P_{\downarrow}$ respectively, induced by their non-zero columns as well as their last column (if it's a zero vector); denote the number of columns of P' and P'' as n' and n'', respectively. Observe that both P' and P'' are still right substochastic matrices. Let $y'_1, \ldots, y'_{n'}$ be the indices in $P_{\uparrow}$ (and thus in P) corresponding to the columns of P', and let $y''_1, \ldots, y''_{n''}$ be the indices in $P_{\downarrow}$ corresponding to the columns of P''. Observe that for any row x, the sum

$$\sum_{\substack{i \geq x \\ j \geq y}} P_{\uparrow}[i,j]$$

is the same for all $y \in (y'_{\ell-1} .. y'_{\ell}]$ for any fixed ℓ.[7] This is because all the columns of $P_{\uparrow}$ in the range $(y'_{\ell-1} .. y'_{\ell} - 1]$ are all zero. A similar observation holds for the matrix $P_{\downarrow}$. For each y, we denote by $\boldsymbol{y'_{\rightarrow}}$ the (unique) smallest index $y'_{\ell} \in \{y'_1, \ldots, y'_{n'}\}$ no smaller than y, and denote by $\boldsymbol{y''_{\rightarrow}}$ the smallest index $y''_{\ell} \in \{y''_1, \ldots, y''_{n''}\}$ no smaller than y.

We recursively compute emulators H' and H'' for P' and P'' respectively. Denote the row and column vertices of H' as $r'[x]$ and $c'[y]$ where x and y range over the rows and columns of P', respectively; similarly denote the row and column vertices of H'' as $r''[x]$ and $c''[y]$ respectively. Take the row vertices of H' and H'' as the row vertices of H, respecting the indices of P. For each column y of P we add a column vertex $c[y]$ to H; connect vertex $c''[y''_{\rightarrow}]$ of H'' to $c[y]$ with an edge of weight zero; and connect vertex $c'[y'_{\rightarrow}]$ of H' to $c[y]$ with an edge of weight

$$\sum_{\substack{i > \lfloor n/2 \rfloor \\ j \geq y}} P[i,j].$$

Thus, since for each pair of indices (x,y) of P one has

$$d_{H'}[r[x], c'[y]] = \sum_{\substack{x \leq i \leq \lfloor n/2 \rfloor \\ j \geq y}} P_{\uparrow}[i,j] \quad \text{and} \quad d_{H''}[r[x], c''[y]] = \sum_{\substack{i \geq x \\ j \geq y}} P_{\downarrow}[i,j],$$

we have that

$$d_H[r[x], c[y]] = \sum_{\substack{i \geq x \\ j \geq y}} P[i,j]$$

[7] For simplicity, one assumes $y'_0 = y''_0 = -\infty$.

as desired. This completes the description of the construction of the emulator for square right substochastic matrices. Observe that the obtained emulator is a directed acyclic graph (dag) with a single path from each row vertex to each column vertex.

Size analysis. We next analyze the size of the emulator. Row vertices are created only at the base of the recursion and there are n of them in total. The number of edges and non-row vertices $N(n)$ satisfies the recurrence $N(n) \leq 2N(n/2) + O(\alpha n)$, so there are overall $O(n \log n)$ vertices and edges in the emulator when the number of columns is linear to the number of rows n.

Time analysis. As for the construction time, the only operation in the recursive procedure that does not clearly require constant time is the computation of the edge weights. However, it is not hard to see that computing all the edge weights of the form

$$w_y := \sum_{\substack{i>\lfloor n/2 \rfloor \\ j \geq y}} P[i,j]$$

in a single recursive call can be done in linear time. This is because w_y is precisely the weight of the unique path in the emulator H'' from the first row vertex to the y'th column vertex. We can therefore simply maintain (with no asymptotic overhead), for every recursively built emulator, the distances from its first row vertex to all its column vertices.

Making weights non-negative. The weight of every edge in our emulator for right substochastic matrix P is non-negative, but the vectors U and V might contain negative entries; as a result the obtained emulator for M might contain negative edges even though every entry in M is non-negative. This can be fixed by a standard reweighting strategy using a *price function*. For the sake of computing the price function $\phi(\cdot)$, add a super-source s and connect s to each row vertex $r[x]$ of the emulator of M by an edge of weight zero. Compute the single-source shortest path tree rooted at s; since our emulator for P is a dag, this can be done in time linear in the size of the emulator. Take the shortest path distances $d(s,\cdot)$ as the price function $\phi(\cdot)$, and let the *reduced weight* $w_\phi(uv)$ of edge uv be $w(uv) + \phi(u) - \phi(v)$. Finally increase the weight of every incoming edge to $c[y]$ by the amount $\phi(y)$. Now all modified weights are non-negative, and all shortest path distances from $r[x]$ to $c[y]$ are preserved because $\phi(r[x]) = 0$ holds for all row vertices $r[x]$.

2.2 The non-bipartite case

Recall that the matrices that capture pairwise distances among consecutive vertices on the boundary of a single face in a planar graph are not (unit-)Monge. In Abboud *et al.* [4] this is handled by decomposing such a matrix recursively into (unit-)Monge submatrices, which incurs a logarithmic overhead. We show how to avoid this logarithmic overhead by constructing emulators for triangular (unit-)Monge matrices. Let $v_1, v_2, \ldots, v_n$ be vertices on the boundary of a single face. Let M be an $n \times n$ matrix such that $M[i,j]$ is the distance from v_i to v_j. For any quadruple (i,i',j,j') satisfying $i < i' < j < j'$, the shortest v_i-to-v_j path must intersect the shortest $v_{i'}$-to-$v_{j'}$ path. Therefore, $M[i,j] + M[i',j'] \geq M[i,j'] + M[i',j]$. Unfortunately, this does not necessarily hold for an arbitrary quadruple. We can, however, define two $n \times n$ unit-Monge matrices M_l and M_u, such that

$$M_l[i,j] = M[i,j] \qquad \text{for all } n \geq i > j \geq 1, \text{ and}$$
$$M_u[i,j] = M[i,j] \qquad \text{for all } 1 \leq i < j \leq n.$$

The other elements of M_l and M_u, as shown by the following lemma, can be (implicitly) filled in so that both matrices are unit-Monge.

► **Lemma 5.** *Let M be a triangular unit Monge matrix. The undefined values of M can be implicitly filled so that M is a $n \times n$ unit Monge matrix.*

Proof. We prove the lemma for a lower triangular matrix M ($n \geq i \geq j \geq 1$). The proof for an upper triangular matrix is similar. For $j > i$ define $M[i,j] = M[j,j] - \sum_{k=i}^{j-1}(M[k+1,k] - M[k,k])$. This choice guarantees that the matrix P for the completed M is zero wherever an undefined element of M is involved. Then, for $j = i+1$,

$$M[i+1,j] - M[i,j] = M[i+1,i+1] - M[i+1,i+1] + M[i+1,i] - M[i,i] = M[i+1,i] - M[i,i],$$

whereas for $j > i+1$ it also holds that,

$$M[i+1,j] - M[i,j] = M[j,j] - \sum_{k=i+1}^{j-1} M[k+1,k] - M[k,k] - M[j,j] + \sum_{k=i}^{j-1} M[k+1,k] - M[k,k] =$$

$$= M[i+1,i] - M[i,i].$$

It follows that for all $i < j$, $M[i+1,j] - M[i,j] = M[i+1,j+1] - M[i,j+1] = M[i+1,i] - M[i,i]$, so the unit Monge property holds for $i < j$. The unit Monge property clearly holds for $i > j$, by definition of lower triangular matrix. It remains to verify that the unit Monge property holds for $i = j$. In that case the Monge inequality $M[i+1,j] - M[i,j] \leq M[i+1,j+1] - M[i,j+1]$ holds because $M[i+1,i] - M[i,i] \leq M[i+1,i+1] - M[i+1,i+1] + M[i+1,i] - M[i,i] = M[i+1,i] - M[i,i]$.

Note that we do not need to explicitly modify M (i.e. compute the artificial values of M). Instead we use the fact that, by our choice of the values that we filled, an element of the matrix P is zero wherever an undefined value of M is involved. ◄

We would like to use an emulator for M_l and an emulator for M_u, but we need to eliminate paths from r_i to c_j for $i < j$ in M_l, and for $i > j$ in M_u. To this end we modify the construction of the emulator. For ease of presentation, we describe the construction for an $(n/2) \times n$ right substochastic matrix P in which we want the distance from c_i to r_j to be infinite for $i > j$. It is easy to modify the construction to work for $(n/2) \times \alpha n$ matrices, for $2n \times \alpha n$ matrices, or for the case where the infinite distances are for $i < j$. The elements of P are classified into two types: an element $P[i,j]$ of P is ***artificial*** if $i > j$ and ***original*** otherwise (that is, the i'th row of P begins with a prefix of $i-1$ artificial elements followed by a suffix of original elements). In the following recursive algorithm, the type of elements is always defined with respect to the full input matrix P (at the top level of the recursion).

We divide the $(n/2) \times n$ matrix P into two $(n/4) \times n$ matrices $P_\uparrow$ and $P_\downarrow$. Both $P_\uparrow$ and $P_\downarrow$ have possibly empty prefix of columns containing just artificial elements (category I), then an infix of columns containing both original and artificial elements (category II), and finally a suffix of columns containing just original elements (category III). Let P' be the submatrix of $P_\uparrow$ induced by all the columns of category II and by the columns of category III that are not entirely zero. Define submatrix P'' of $P_\downarrow$ similarly. Note that, by definition of artificial elements (with respect to the original top-level matrix), the number of columns of category II is bounded by the number of rows in $P_\uparrow$ and $P_\downarrow$. Also, by definition of right substochastic matrix, the number of columns of category III that are not entirely zero is also bounded by the number of rows of $P_\uparrow$ and $P_\downarrow$. Hence, both P' and P'' are $(n/4) \times (n/2)$ matrices for which we can recursively compute emulators H' and H'', respectively.

We construct the emulator H of P from the emulators H' and H'' as follows. For each column y of P we add a vertex $c[y]$ to H. If column y in $P_\uparrow$ consists of just artificial elements (category I), we do nothing. If column y in $P_\uparrow$ is of category II, then column y is also present in P', so it

has a corresponding column vertex $c'[y]$ in H'. Connect $c'[y]$ to $c[y]$. If column y is of category III, connect vertex $c'[y'_{\rightarrow}]$ of H' to $c[y]$ with an edge of weight

$$\sum_{\substack{i>\lfloor n/4 \rfloor \\ j \geq y}} P[i,j].$$

Treat all the columns in $P_{\downarrow}$ similarly, except when column y is of category III, connect vertex $c''[y''_{\rightarrow}]$ of H'' to $c[y]$ with an edge of weight zero. (For the definition of $y'_{\rightarrow}$ and $y''_{\rightarrow}$ see the proof of the bipartite case.) An easy inductive proof shows that in this construction r_i is connected to c_j if and only if $i < j$.

The size of the emulator again satisfies the recurrence $N(n) \leq 2N(n/2) + O(n)$, and therefore is of $O(n \log n)$ overall. This concludes the proof of Lemma 3.

3 Distance Emulators for Planar Graphs

In this section we explain how to incorporate the unit-Monge emulators of Lemma 3 into the construction of planar graph emulators. The construction follows closely to the encoding scheme of Abboud *et al.* [4]. The crucial difference is that their unit-Monge matrices are represented by a compact non-graphical encoding, which we replace by the distance emulator of Lemma 3. There are additional differences that stem from the fact that Abboud *et al.* concentrated on the size of the encoding, and did not consider the construction time. To achieve efficient construction time we have to modify the construction slightly. These modifications are not directly related to the unit-Monge distance emulators. We will describe the construction algorithm without going into details, which were treated in Abboud *et al.* [4]. Instead, we attempt to give some intuition and describe the main components of the construction and their properties. See Appendix A for a self-contained detailed description.

Compact representations of distances in planar graphs have been achieved in many previous works by decomposing the graph using small cycle separators and exploiting the Monge property (not the unit-Monge property) to compactly store the distances among the vertices of the separators. The main obstacle in exploiting the unit-Monge property using this approach is that small cycle separators do not necessarily exist in planar graphs with face of unbounded size. In weighted graphs this difficulty is overcome by triangulating the graph with edges with sufficiently large weights that do not affect distances. In unweighted graphs, however, this does not work.

Slices. To overcome the above difficulty, Abboud *et al.* showed that any unweighted planar graph contains a family of nested subgraphs (called ***components***) with some desirable properties. This nested family can be represented by a tree $\mathscr{K}$, called the ***component tree***, in which the ancestry relation corresponds to enclosure in G. Each component K in $\mathscr{K}$ is a subgraph of G that is enclosed by a simple cycle ∂K, called the ***boundary cycle*** of K. The root component is the entire graph G, and its boundary cycle is the infinite face of G. A ***slice*** K° is defined as the subgraph of G induced by all faces of G that are enclosed by component K, and not enclosed by any descendant of K in $\mathscr{K}$. The boundary cycle ∂K is called the ***external hole*** or the ***boundary*** of K°. The boundary cycles of the children of K in $\mathscr{K}$ are called the ***internal holes*** of K°. Note that a slice may have many internal holes. Abboud *et al.* showed that, for any choice of $w \in [n]$, there exists a component tree $\mathscr{K}$ such that the total number of vertices in the boundaries of all the slices is $O(n/w)$ [4, Lemma 3.1]. For completeness, we present this proof in Appendix A.1.

Regions. Constructing distance emulator for each slice separately is not good enough naïvely, as there could potentially be $\Omega(n/w)$ holes in a slice, and we cannot afford to store distances between each pair of holes, even if we use the unit-Monge property. To overcome this, each

slice K° is further decomposed recursively into smaller subgraphs called ***regions***. A *Jordan cycle separator* is a closed curve in the plane that intersects the embedding only at $O(w)$ vertices. A slice K° is recursively decomposed in a process described by a binary ***decomposition tree*** $\mathscr{R}_K$; each node R of the tree $\mathscr{R}_K$ is a region of K°, with the root being the entire slice K°. There is a unique Jordan cycle separator C_R corresponding to each internal node R of $\mathscr{R}_K$. We abuse the terminology by referring to an internal hole of slice K° lying completely inside region R as a *hole* of R. We say that a Jordan curve C_R *crosses* a hole H of R if C_R intersects both inside and outside of ∂H. All Jordan cycle separators discussed here are mutually non-crossing (but may share subcurves) and each crosses a hole at most twice.

We next describe how the separators C_R are chosen. Let K be a component, and let K' be the child component of K in $\mathscr{K}$ with the largest number of vertices. We designate the hole $\partial K'$ of the slice K° as the ***heavy*** hole of K°.[8] For any region R of K°, let $\boldsymbol{R^\bullet}$ denote the union of R and all the subgraphs of G enclosed by the internal holes of R except for the heavy hole of K° (in other words, $R^\bullet$ is the region R with all its holes except the heavy hole "filled in"). For example, given a slice K°, one has $(K^\circ)^\bullet = K \setminus int(\partial H^*)$, where $int(\partial H^*)$ is the interior of the heavy hole H^* of K° (if any). Abboud *et al.* [4, Lemma 3.4] proved that for any region R one of the following must hold (and can be applied in time linear in the size of R):

- One can find a small Jordan cycle separator C_R that crosses no holes of R and is balanced with respect to the number of terminals in $R^\bullet$. That is, C_R encloses a constant fraction of the weight with respect to a weight function assigning each terminal in R unit weight and each face corresponding to a non-heavy hole of R weight equal to the number of terminals enclosed (in $R^\bullet$) by that hole. Such a separator is called a ***good*** separator.
- There exists a hole H such that one can *recursively* separate R with small Jordan cycle separators, each of which crosses H exactly twice, crosses no other hole, and is balanced with respect to the number of vertices of ∂H in the region R. It is further guaranteed that each resulting region R' along this recursion satisfies that $(R')^\bullet$ contains at most half the terminals in $R^\bullet$. The hole H is called a ***disposable hole***, and the recursive process is called the ***hole elimination*** process. The hole elimination process terminates if a resulting region R' either contains a constant number of vertices of the disposable hole ∂H, or if $(R')^\bullet$ contains no terminals. Thus, the hole elimination process is represented by an internal subtree of $\mathscr{R}_K$ of height $O(\log n)$.

We begin with the region consisting of the entire slice K°. If a good separator is found, we use it to separate a region R into two subregions, each containing at most a constant fraction of the terminals in $R^\bullet$. If no good separator is found then we use the hole elimination process on a disposable hole H. We stop the decomposition as soon as a region contains at most one hole (other than the heavy hole) or at most one terminal. Thus, the overall height of $\mathscr{R}_K$ is $O(\log k \cdot \log n)$. We mentioned that it is guaranteed that each Jordan cycle separator used crosses a hole at most twice. This implies that, within each region R, the vertices of each hole ∂H of K° that belong to R can be decomposed into $O(\log k \cdot \log n)$ intervals called ***∂H-intervals***; each Jordan cycle separator increases the number of such intervals in each subregion by at most one. We refer to those intervals that belong to the boundary cycle ∂K of K° or to the heavy hole ∂H^* of K° as ***boundary intervals***.

[8] This is the main difference compared with the encoding scheme of Abboud *et al.* [4]. We will see the benefit of this technicality in Section 3.3.

3.1 The construction

We are now ready to describe the construction of the distance emulator. We will build the distance emulator of G from the leaves of the component tree $\mathscr{K}$ to the root (which is the whole graph G). For each component K in $\mathscr{K}$, we will associate an emulator that preserves the following distances.

- *terminal-to-terminal* distances in K between any two terminals in K.
- *terminal-to-boundary* distances in K between any terminal in K and vertices of boundary ∂K.

We emphasize that the distances being preserved are those in K, not K°. The distance emulator for G can be recovered from the emulator associates with the root of the component tree $\mathscr{K}$.

We now describe the construction of the emulator associated with component K, assuming all the emulators for the children of K in $\mathscr{K}$ have been constructed. The emulator of K is constructed by adding the following edges and unit-Monge emulators from Lemma 3 to the emulators obtained from the children of K.

1. Add unit-Monge emulators representing the distances in K° between all pairs of vertices on the *boundary* ∂K of K or on the *heavy hole* ∂H^* of K.
2. For each region R in the decomposition tree $\mathscr{R}_K$ of the slice K°, add the following edges and the unit-Monge emulators according to the type of R.
   - **(A)** If R is an internal node of $\mathscr{R}_K$ with separator C_R (either good or hole-eliminating):
     - **i.** Add an edge between each *terminal* in $R^\bullet$ and each vertex on the *separator* C_R. The weight of each edge is the distance in $R^\bullet$ between the two vertices.
     - **ii.** For each of the two subregions R_1 and R_2 of R, add a unit-Monge emulator representing the distance in $R_i^\bullet$ between the *separator* C_R and each *boundary interval* of R_i.
     - **iii.** If R corresponds to a cycle separator C_R eliminating a disposable hole ∂H:
       - **(a)** If R is the root of the subtree of $\mathscr{R}_K$ representing the hole-elimination process of a *hole* H, add a unit-Monge emulator representing the distances between ∂H and each *boundary interval* of R. The distances are in the subgraph obtained from $R^\bullet$ by removing the subgraph strictly enclosed by ∂H.
       - **(b)** For each of the two subregions R_1 and R_2 of R, add a unit-Monge emulator representing the distance in $R_i^\bullet$ between the *separator* C_R and each *∂H-interval* of R_i.
   - **(B)** If R is a leaf of $\mathscr{R}_K$ and all terminals in $R^\bullet$ are enclosed by a single non-heavy hole ∂H:
     - **i.** Add a unit-Monge emulator representing the distances between ∂H and each *boundary interval* of R. The distances are in the subgraph obtained from $R^\bullet$ by removing the subgraph strictly enclosed by ∂H.
   - **(C)** If R is a leaf of $\mathscr{R}_K$ that contains exactly one terminal t:
     - **i.** Add an edge between the only *terminal* t and each vertex on each *boundary interval* of R. The weight of each edge is the distance in $R^\bullet$ between the two vertices.

The proof of correctness is essentially the same as that in Abboud *et al.* [4, Lemma 3.6].

3.2 Space analysis

The $O(\sqrt{k \cdot n} \log^3 n)$ space analysis is essentially the same as in Abboud *et al.* [4]. The main difference in the algorithm is the introduction of the heavy hole as part of the boundary of a slice. This will allow an efficient construction time and does not affect the construction space because it increases the total size of the boundary by a constant factor.

Since a boundary cycle appears at most twice as a boundary of a slice (once as the external boundary and once as the heavy hole), the total size for all unit-Monge emulators for boundary-to-boundary distance (1) is $O((n/w)\log n)$.

We next bound the total space for terminal-to-separator distances (type 2(A)i) over all slices. Whenever a terminal stores its distance to a separator, the total number of terminals in its region $R^\bullet$ decreases by a constant factor within $O(\log n)$ levels of the decomposition tree (either immediately if the separator is a good separator, or within $O(\log n)$ levels of the hole elimination process), so this can happen $O(\log^2 n)$ times per terminal. Since each separator has size $O(w)$, the total space for representing distances of type 2(A)i is $O(kw\log^2 n)$.

The space required to store a single unit-Monge distance emulator representing distances between a separator C_R and a boundary interval b (type 2(A)ii) is $O(|C_R|+|b|\log|b|)$ (The rows of such an emulator correspond to the vertices of b, and the columns to vertices of C_R. Each row in the corresponding right sub-stochastic matrix has at most a single non-zero entry). We note the following facts: (i) The size of any separator C_R is $O(w)$. (ii) The depth of the recursion within each slice is $O(\log^2 n)$. (iii) There are at most k regions at each level of the recursion. (iv) The total number of boundary intervals in all regions at a specific recursive level is within a constant factor from the number of regions at that level, i.e., $O(k)$. Thus, the total size of all boundary intervals at a fixed level of the recursion is at most $O(k)$ plus the size of the boundary of the slice (boundary intervals are internally disjoint, and there are $O(k)$ intervals in each region). (v) The total boundary size of all slices combined is $O(n/w)$. By the above facts, the total space for type 2(A)ii emulators is $O((kw+(k+n/w)\log n)\log^2 n) = O((kw+n/w)\log^3 n)$.

Hole-to-boundary distances (types 2(A)iii(a) and 2(B)i) are stored for at most one hole in each region along the recursion, and require $O(|\partial H|+|b|\log|b|)$ space. By the same arguments as above, and since the total size of hole boundaries ∂H is bounded by the total size of slice boundaries, the total space for types 2(A)iii(a) and 2(B)i emulators is $O((k+n/w)\log^3 n)$.

The space for separator-to-hole emulators (type 2(A)iii(b)) is bounded by the separator-to-boundary distances (type 2(A)ii).

To bound the number of terminal-to-boundary edges (type 2(C)i) note that each terminal stores distances to boundary intervals that are internally disjoint for distinct terminals. Since the number of boundary terminals in a region is $O(\log n)$, the total space for type 2(C)i distances is $O(k\log n + n/w)$.

The overall space required is thus $O((kw+n/w)\log^3 n)$; setting $w=\sqrt{k\cdot n}$ gives the bound $O(\sqrt{k\cdot n}\log^3 n)$.

3.3 Time analysis

Recall that Lemma 3 assumes that the input is an $n\times n$ unit-Monge matrix represented in $O(n\log n)$ bits as two arrays U and V, as well as an array P, specifying, for each row i of the right substochastic $2(n-1)\times(n-1)$ matrix P, the index of the (at most) single one entry in that row (see Lemma 4). Lemma 3 will be invoked on various $x\times x$ unit-Monge matrices that represent distances among a set of x vertices on the boundary of a single face of a planar graph G with n vertices (or sometimes, on two sets of vertices on two distinct faces).

▶ **Lemma 6.** *Let G be a planar graph with n nodes. Let f be a face in G. One can compute the desired representation of the unit-Monge matrix M representing distances among the vertices of f in $O(n)$ time.*

Proof. We tweak the linear-time multiple source shortest paths (MSSP) algorithm of Eisentat and Klein [29]. Along its execution, the MSSP algorithm maintains the shortest path trees rooted at each vertex of a single distinguished face f, one after the other. This is done by transforming the

shortest path tree rooted at some vertex v_i to the shortest path tree rooted at the next vertex v_{i+1} in cyclic order along the distinguished face f. Let $d_i(u)$ denote the distance from the current root v_i to v. The slack of an arc uw with respect to root v_i is defined as $slack_i(uw) := c(uw)+d_i(u)-d_i(w)$. Thus, for all arcs in the shortest path tree the slack is zero, and for all arcs not in the shortest path tree the slack is non-negative. Eisenstat and Klein show that throughout the entire execution of the algorithm there are $O(n)$ changes to the slacks of edges. Their algorithm explicitly maintains and updates the slacks of all edges as the root moves along the distinguished face. By definition of the matrix P,

$$P[i,j] \;=\; M[i+1,j+1]+M[i,j]-M[i+1,j]-M[i,j+1].$$

Consider an edge v_jv_{j+1} along the distinguished face. Its slack with respect to root v_i is $1+M[i,j]-M[i,j+1]$. Its slack with respect to root v_{i+1} is $1+M[i+1,j]-M[i+1,j+1]$. Therefore, $P[i,j] = slack_i(v_jv_{j+1})-slack_{i+1}(v_jv_{j+1})$. It follows that one can keep track of the nonzero entries of P by keeping track of the edges of the distinguished face whose slack changes. The entries of the arrays U and V can be obtained in total $O(n)$ time since they only depend on the distances from a single vertex of f. ◀

We can now analyze the construction time. The boundary-to-boundary distances (type 1) in a single slice are computed by a constant number of invocations of MSSP (Lemma 6) on $K^\bullet$. We then compute the emulator for each unit-Monge matrix on x vertices in $O(x\log x)$ time using Lemma 3. For the total MSSP time, we need to bound the total size of the regions $R^\bullet$ on which we invoke Lemma 6. This is where we use the fact that the heavy hole of a slice K° contains at least half the vertices of K. Since $K^\bullet$ does not include the heavy hole of K, the standard heavy path argument implies that the number of slices in which a vertex v participates in invocations of the MSSP algorithm is at most 1 plus $O(\log n)$ (the number of non-heavy holes v belongs to). Thus, each vertex v participates in invocations of MSSP for $O(\log n)$ slices. In each invocation of MSSP each vertex v is charged $O(\log n)$ time, so the total time for all MSSP invocations is $O(n\log^2 n)$. To bound the total time of the invocations of Lemma 3, note that each boundary cycle appears at most twice as a boundary of a slice (once as the external boundary and once as the heavy hole). Hence, the total time for all invocations of Lemma 3 is $O((n/w)\log n)$. Overall we get that the total time to compute all the emulators of type 1 is $O(n\log^2 n+(n/w)\log n) = O(n\log^2 n)$.

The terminal-to-separator distances (type 2(A)i) are computed by running MSSP twice, once for the interior of the separator C_R in $R^\bullet$ and once for the exterior of C_R in $R^\bullet$. This takes $O(|R^\bullet|\log|R^\bullet|)$ time. Then, we can report the distance, within each of these two subgraphs of $R^\bullet$, from each vertex of C_R to any other vertex in $O(\log n)$ time. Next, for each terminal t in $R^\bullet$, we compute the distance from t to all w vertices of C_R by running FR-Dijkstra [33] on the graph consisting of the edges between t and the vertices of C_R as well as the two complete graphs representing the distances among the vertices of C_R in the interior and exterior. This takes $O(w\log^2 w\log n)$ time. The $O(w\log^2 w)$ term is the running time of FR-Dijkstra and the additional $O(\log n)$ factor is because each distance accesses by FR-Dijkstra is retrieved on-the-fly from the MSSP data structure.

We analyze separately the total cost of all the MSSP computations and the total cost of computing the edge weights. For the latter, we have argued in the analysis of the emulator's size that each terminal participates in the computation of edges of type 2(A)i in $O(\log^2 n)$ regions along the entire construction. Therefore, the total time to compute all such edges is $O(kw\log^2 w\log^3 n)$. For the total MSSP time, we need to bound the total size of the regions $R^\bullet$ on which we run MSSP. This is done in a similar manner to the bound on the time for MSSP invocations for type 1 above, using the standard heavy path argument. We argued that the number of slices in which a vertex v participates in invocations of the MSSP algorithm is at most 1 plus $O(\log n)$ (the number

of non-heavy holes v belongs to). Thus, each vertex v participates in invocations of MSSP for $O(\log n)$ slices. Since the depth of the decomposition tree of a slice is $O(\log^2 n)$, each vertex participates in at most $O(\log^3 n)$ invocations of MSSP in the computation of edges of type 2(A)i. Each vertex is charged $O(\log n)$ time in each invocation of MSSP it participates in, so the total time for all invocations of MSSP in the process of computing edges of type 2(A)i is $O(n \log^4 n)$.

To bound the time required to compute the emulators of type 2(A)ii (separator-to-boundary distances), note that the appropriate unit-Monge matrices can be computed by MSSP invocations within the same time bounds analyzed for the MSSP invocations for terminal-to-separator edges above. Hence, we only need to account for the additional time spent on constructing the emulators by invocations of Lemma 3, which is linear in the total size of these emulators, which we have already shown above to be $O((kw + n/w)\log^3 n)$.

The analysis for the time required for the hole-to-boundary emulators (types 2(A)iii(a) and 2(B)i) uses the same arguments as the analysis for type 2(A)ii. Again, the MSSP time is $O(n \log^4 n)$, and the time for invocations of Lemma 3 is linear in the size of these emulators, which is $O((k + n/w)\log^3 n)$.

The time to compute the separator-to-hole emulators (type 2(A)iii(b)) is bounded by the time to compute the separator-to-boundary emulators (type 2(A)ii).

To compute the terminal-to-boundary edges we again need to invoke MSSP once on $R^\bullet$ for each leaf region R, and then query the distance in $O(\log n)$ time per distance. The time for MSSP is dominated by the MSSP time accounted for in type 2(A)ii emulators, and since we have shown that the total number of such edges is $O(k + n/w)$, the total time to compute them is $O((k + n/w)\log n)$.

The overall construction time is therefore dominated by the $O(n \log^4 n)$ term. Combined with the space analysis of Section 3.2, we establish Theorem 1.

Acknowledgment. The authors would like to thank Timothy Chan, Jeff Erickson, Sariel Har-Peled, and Yipu Wang for helpful discussions. The first author express special thanks to 施鴻逸 (Hong-Yi Shih) for discussion back in NTU in the early days that made this paper possible.

References

1 Amir Abboud and Greg Bodwin. Error amplification for pairwise spanner lower bounds. In *27th SODA*, pages 841–854, 2016.

2 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017.

3 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *28th SODA*, pages 568–576, 2017.

4 Amir Abboud, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal compression for the planar graph metric. In *29th SODA*, pages 530–549, 2018.

5 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

6 Alok Aggarwal and James K. Park. Notes on searching in multidimensional monotone arrays (preliminary version). In *29th FOCS*, pages 497–512, 1988.

7 Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *3rd SoCG*, pages 278–290, 1987.

8 Noga Alon. Testing subgraphs in large graphs. *Random Struct. Algorithms*, 21(3-4):359–370, 2002.

9 Stephen Alstrup, Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Ely Porat. Sublinear Distance Labeling. In *24th ESA*, pages 5:1–5:15, 2016.

10 Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In *27th SODA*, pages 338–350, 2016.

11 Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics. *Unpublished Manuscript, available from http://www.math.ucdavis.edu/˜abasu/papers/SPR.pdf*, 2008.

12 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β)-spanners and purely additive spanners. In *16th SODA*, pages 672–681, 2005.

13 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *21st CPM*, pages 138–150, 2010.

14 Greg Bodwin. Linear size distance preservers. In *28th SODA*, pages 600–615, 2017.

15 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. In *27th SODA*, pages 855–872, 2016.

16 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Mathematics*, 19(4):1029–1055, 2005.

17 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *52nd FOCS*, pages 170–179, 2011.

18 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. In *51st FOCS*, pages 601–610, 2010.

19 Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012.

20 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *28th SODA*, pages 2143–2152, 2017.

21 Hubert T.-H. Chan, Donglin Xia, Goran Konjevod, and Andréa W. Richa. A tight lower bound for the steiner point removal problem on trees. In *9th APPROX*, pages 70–81, 2006.

22 Hsien-Chih Chang and Hsueh-I Lu. Computing the girth of a planar graph in linear time. *SIAM Journal on Computing*, 42(3):1077–1094, 2013.

23 Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately - lower and upper bounds. In *43rd ICALP*, pages 131:1–131:14, 2016.

24 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proc. 12th Symp. Discrete Algorithms*, pages 506–515, 2001.

25 Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. In *28th SODA*, pages 534–553, 2017.

26 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.

27 Maxime Crochemore, Gad. M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32:1654–1673, 2003.

28 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

29 David Eisenstat and Philip N Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *45th STOC*, pages 735–744, 2013.

30 Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

31 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016.

32 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing*, 43(4):1239–1262, 2014.

33 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006.

34 Thomas A. Feo and J. Scott Provan. Delta-wye transformations and the efficient reduction of two-terminal planar graphs. *Oper. Res.*, 41(3):572–582, 1993.

35 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

36 Paweł Gawrychowski, Adrian Kosowski, and Przemysław Uznański. Sublinear-space distance labeling using hubs. In *30th DISC*, pages 230–242, 2016.

37 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in Monge matrices are equivalent to predecessor search. In *42nd ICALP*, pages 580–592, 2015.

38 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th ESA*, pages 44:1–44:14, 2017.

39 Gramoz Goranci and Harald Räcke. Vertex sparsification in trees. In *14th WAOA*, pages 103–115, 2016.

40 Ronald L Graham and Henry O Pollak. On embedding graphs in squashed cubes. In *Graph theory and applications*, volume 303, pages 99–110. 1972.

41 Anupam Gupta. Steiner points in tree metrics don't (really) help. In *12th SODA*, pages 220–227, 2001.

42 Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. A unified algorithm for accelerating edit-distance via text-compression. *Algorithmica*, 65:339–353, 2013.

43 Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. In *16th SWAT*, pages 26:1–26:12, 2018.

44 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *43rd STOC*, pages 313–322, 2011.

45 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *15th ISAAC*, pages 558–568, 2004.

46 Sampath Kannan, Moni Naor, and Steven Rudich. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

47 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *23rd SODA*, pages 338–355, 2012.

48 M. M. Klawe and D J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM Journal Discrete Math.*, 3(1):81–97, 1990.

49 Philip Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n \lg^2 n)$-time algorithm. *ACM Transactions on Algorithms*, 6(2):2–13, 2010.

50 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 505–514, 2013.

51 Lukasz Kowalik and Maciej Kurowski. Short path queries in planar graphs in constant time. In *35th STOC*, pages 143–148, 2003.

52 Robert Krauthgamer, Huy L Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014.

53 Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. Preprint, July 2017.

54 Jakub Lacki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd FOCS*, pages 599–608, 2012.

55 J. W. Moon. On minimal n-universal graphs. *Glasgow Mathematical Journal*, 7(1):32–33, 1965.

56 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd SODA*, pages 209–222, 2012.

57 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. 38th Annual Symposium on Foundations of Computer Science*, pages 118–126, 1997.

58 Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.

59 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *17th SODA*, pages 802–809, 2006.

60 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications. *Arxiv 0707.3619*, 2007.

61 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. In *21st SODA*, pages 1287–1296, 2010.

62 György Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

63 Oren Weimann and Raphael Yuster. Computing the girth of a planar graph in $O(n \log n)$ time. *SIAM J. Discrete Math.*, 24(2):609–616, 2010.

64 Peter M Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.

65 David P Woodruff. Lower bounds for additive spanners, emulators, and more. In *47th FOCS*, pages 389–398, 2006.

66 Christian Wulff-Nilsen. Wiener index and diameter of a planar graph in subquadratic time. In *25th EuroCG*, pages 25–28, 2009.

67 Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time. *Computational Geometry*, 46(7):831–838, 2013.

A Missing Details

The construction follows closely to the sublinear-size encoding scheme of undirected unweighted planar graphs described by Abboud *et al.* [4], while carefully replacing each encoding of unit-Monge matrix with the corresponding distance emulator guaranteed by Lemma 3 and Section 2. One of the main difference here is that Abboud *et al.* did not focus on or analyze the construction time. We have introduced some changes to their construction in order to reduce the construction time of the emulator. We will emphasize the changes in the subsequent subsections.

A.1 Slices

The ***radial graph*** $G^\diamond$ of a plane graph G is the face-vertex incidence graph of G; the vertex set of $G^\diamond$ consists of the vertices and faces of G, and the edge set of $G^\diamond$ corresponds to all the vertex-face incidences in G. The radial graph $G^\diamond$ is also a plane graph with its embedding inherited from G. Notice that for any plane graph G and its dual G^*, the radial graph $G^\diamond$ is identical.

Perform a breadth-first search from the unbounded face of G in $G^\diamond$. The ***level*** of a vertex or face in G is defined to be its depth in the breadth-first search tree of $G^\diamond$. It is well-known [34, 50] that the odd levels (which correspond to the vertices in G) forms a laminar family of edge-disjoint simple cycles. Formally, let $\mathscr{K}_{\geq i}$ be the collection of connected components of subgraph of the dual graph G^* induced by faces of levels at least $2i$; for simplicity we call each connected components in $\mathscr{K}_{\geq i}$ as ***component*** at level i. The collection of components at all levels and their containment relationship forms a tree structure, which we refer to as the ***component tree*** $\mathscr{K}$.[9] Component K' is a descendant of another component K if the faces of K' are a subset of the faces of K. The root of the component tree corresponds to the component of G that is the union of all faces in G except for the unbounded face. The boundary of a component K always forms a simple cycle in G; we often refer it as the ***boundary cycle*** ∂K of K.

A *slice* of G is defined to be the subgraph of G induced by all faces of G enclosed by the boundary cycle of some component at level i and not strictly enclosed by the boundary cycles of any component at level j for some $j > i$. In this paper we will only consider slices of the following form: For a fixed level ℓ and some integer ***width*** w, consider any slice defined by some component at the levels of the form $\ell_\alpha := \ell + \alpha w$ for all integer $\alpha \geq 0$, as well as level 0 (which the component corresponds to the whole graph G). For each component K at level ℓ_α in the component tree $\mathscr{K}$ one can define ***slice*** K° between component K and all the descendants of K in $\mathscr{K}$ at level $\ell_{\alpha+1}$. Each slice inherits its embedding from G. By definition K° is a subgraph of K. The boundary cycle of K is also the boundary cycle of the unbounded face of the slice K°, called the ***external hole*** or ***boundary*** of K°; and the boundary cycles of components at level $\ell_{\alpha+1}$ in K° are called the ***internal holes*** of K°. Bear in mind that the definition of external and internal holes depends on the slice we are currently in; an internal hole of the slice K° is the external hole of some slice K'° for some descendant K' of K.

For the sake of simplicity we assume that the unbounded face of G has length 3 by adding a triangle enclosing the whole graph G, and add a single edge between the triangle and G. Now an averaging argument, together with the length assumption on the unbounded face, guarantees that for any fixed width w there will be a choice of ℓ such that the sum of the length of all the holes in the slices defined is at most $O(n/w)$ [4, Lemma 3.1].

[9] To describe the emulator construction we will be using several different trees whose nodes correspond to very different objects. To avoid confusion, we will consistently denote trees whose nodes are subgraphs of G using calligraphy letters, and denote trees whose nodes are vertices of G using normal uppercase letters.

A.2 Regions

Constructing distance emulator directly for each slice is not good enough naïvely, as there could potentially be $\Omega(n)$ holes in a slice. Now the strategy is to decompose the slice K° into smaller pieces called *regions*, such that each region has "small boundary", has at least one terminal, and has at most a constant number of internal holes. The distance information will be stored using the regions, such that for each slice K° the distances *in* K between all terminals and vertices on holes are kept by the distance emulator (we will see shortly in Section 3.1 why it is important to store distances with respect to the whole component K, not just the slice K°).

Each slice K° contains at most w consecutive levels of breadth-first search tree. One can triangulate the slice K° into K°_Δ in a way that the breadth-first search tree on K° starting at an auxiliary vertex representing the unbounded face of K° is consistent with the breadth-first search tree of the radial graph $G^\diamond$ that defines the levels and slices [4, Lemma 3.2], and each hole of K° has an auxiliary vertex connecting to all the boundary vertices on the hole, except for the internal hole that contains more than half of the vertices in K (if any). (Such holes are called ***heavy***; it is straightforward to see that there can be at most one heavy hole in a slice.[10]) More precisely, let $\boldsymbol{T_K}$ be the breadth-first search tree of K°_Δ, then vertex u is a parent of vertex v in T_K if and only if u is a grandparent of v in the breadth-first search tree of $G^\diamond$. All the edges incident to the auxiliary vertex of a hole ∂H belong to T_K if ∂H is external, and exactly one of such edges belongs to T_K if ∂H is internal. One can compute the triangulation K°_Δ and the breadth-first search tree T_K from K° in time linear in the size of K°.

Fix a slice K° for some component K. We now describe a high-level process to decompose K° into regions using fundamental cycles with respect to T_K. Given any subgraph R of the slice K° called a ***region*** starting with $R := K^\circ$, pick an edge e_R in K°_Δ but not in T_K. (We will describe how to pick such an edge in Section A.3.) For each region R, let C_R be the *fundamental cycle* of edge e_R with respect to T_K (i.e., the cycle composed of e_R plus the unique path in T_K between the endpoints of e_R). Each fundamental cycle C_R will be a *Jordan curve* in the plane that separates R into two (potentially trivial) subgraphs that share all the vertices and edges on $C_R \cap R$. We emphasize that by the construction of T_K, cycle C_R only intersects each (external or internal) hole of K° at most twice. Recursively decompose the two subgraphs into regions. Then the regions of K° is defined to be the collection of all the regions constructed from recursion. One can associate a ***decomposition tree*** $\boldsymbol{\mathscr{R}_K}$ of slice K° that encodes the region construction process; each node of the tree $\mathscr{R}_\mathscr{K}$ is a region of K°, and there is a unique fundamental cycle C_R corresponding to each internal node R of $\mathscr{R}_K$. We abuse the terminology by calling an internal hole of slice K° lying completely inside region R as a *hole* of R.

A.3 Good cycles and disposable holes

What is left is to describe the choice of fundamental cycle that corresponds to each internal node of the decomposition tree $\mathscr{R}_K$. For any region R, let $\boldsymbol{R^\bullet}$ denote the union of R and all the subgraphs enclosed by the internal holes of R except for the heavy one; equivalently, $R^\bullet$ is the subgraph of K induced by all the faces of K enclosed by the (weakly-simple) boundary cycle of R, after removing the interior of any hole of R containing more than half of the vertices in K (this technicality was discussed in detail in Section 3.3). For example, given a slice K°, one has $(K^\circ)^\bullet = K \setminus int(\partial H^*)$, where $int(\partial H^*)$ is the interior of the heavy hole of K° (if any).

[10] This is the main difference to the encoding scheme of Abboud *et al.* [4]. We will discuss this technicality in Section 3.3.

Fix a region R in $\mathscr{R}_K$. We say a fundamental cycle C with respect to T_K is ***good*** for region R if C is a balanced separator with respect to number of terminals in $R^\bullet$, and C does not intersect the interior of any hole of R. A hole ∂H of R is ***disposable*** if each subgraph induced on the faces enclosed by the fundamental cycle C of each edge on ∂H contains at most half of the terminals in $R^\bullet$. Abboud *et al.* proved the following structural result that either one can find a good fundamental cycle separator and recurse on the two subgraphs, or there must be a disposable hole, in which case one perform a "hole-elimination" process to remove the disposable hole.

► **Lemma 7** (Abboud *et al.* [4, Lemma 3.4])**.** *Given an arbitrary region R such that $R^\bullet$ contains more than one terminal, there is either a good fundamental cycle for R or a disposable hole in R.*

Now we are ready to describe the construction of $\mathscr{R}_K$. For each region R, if R has at most one hole (other than the heavy hole of the slice K°) and at most one terminal then we stop. Otherwise we apply Lemma 7 on R. If there is a good fundamental cycle C for R then we choose the edge defining C to be the edge e_R associated with R. Let R' and R'' be the two subgraphs induced by the faces of the interior and exterior of C. Attach R with two children R' and R'' in the decomposition tree $\mathscr{R}_K$, and recursively construct the subtrees rooted at R' and R''. If there is a disposable hole ∂H, we cut the region R open along each path in T_K from the root to the vertices of ∂H. Attach a binary tree of height $O(\log n)$ rooted at R in $\mathscr{R}_K$ by iteratively finding balanced fundamental cycle separators with respect to the number of vertices on ∂H among those using the artificial edges incident to the auxiliary vertex in the hole ∂H and decompose the region R into two smaller pieces. The leaves of the attached tree correspond to the regions obtained from cutting the paths in T_K from root to all the vertices of ∂H. Remove all the regions that do not contain any terminals; recursively construct the subtrees rooted at the rest of the regions.

By definition a good fundamental cycle separator decreases the number of terminals in the resulting regions by a constant factor. Similarly, by definition of disposable holes, each of the regions obtained at the end of the hole-elimination process has at most half of the terminals in the original region. Consequently the height of the decomposition tree $\mathscr{R}_K$ is at most $O(\log |T| \cdot \log n)$ for each slice K°.

As we mentioned before, while the boundary of a slice K° must be a simple cycle, that's not the case for the regions in $\mathscr{R}_K$ in general; additionally, the boundary of a region consists of fragments of vertices coming from either the holes (both external and internal) or some fundamental cycle separators. Each fragment that comes from some hole ∂H must be a simple path; we call such fragments ***∂H-intervals***. We refer those intervals that belong to the boundary cycle ∂K of K or to the heavy hole ∂H^* of K° as ***boundary intervals***. Clearly, the region K° has at most two boundary intervals (one is the simple external boundary of K and the other is the simple boundary of the heavy hole of K°). By construction of the spanning tree T_k, whenever a region is separated by a fundamental cycle separator with respect to T_k, the number of boundary intervals increases by at most two. Using the analysis on the recursion height of the decomposition tree $\mathscr{R}_K$, we conclude that for each hole ∂H, there are at most $O(\log |T| \cdot \log n)$ ∂H-intervals on the boundary of R. We will use this fact in Section 3.1 to bound the size of the construction.