Almost-Linear ε-Emulators for Planar Graphs*

Hsien-Chih Chang
Dartmouth College
USA

Robert Krauthgamer†
Weizmann Institute of Science
Israel

Zihan Tan‡
University of Chicago
USA

ABSTRACT

We study vertex sparsification for distances, in the setting of planar graphs with distortion: Given a planar graph G (with edge weights) and a subset of k terminal vertices, the goal is to construct an *ε-emulator*, which is a small planar graph G' that contains the terminals and preserves the distances between the terminals up to factor $1+\varepsilon$.

We design the first ε-emulators for planar graphs of almost-linear size $k^{1+o(1)}/\mathrm{poly}\,\varepsilon$. In terms of k, this is a dramatic improvement over the previous quadratic upper bound of Cheung, Goranci and Henzinger [ICALP 2016], and breaks below known quadratic lower bounds for exact emulators (the case when $\varepsilon = 0$). Moreover, our emulators can be computed in near-linear time, with applications to fast $(1+\varepsilon)$-approximation algorithms for basic optimization problems on planar graphs such as minimum (s,t)-cut and diameter.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; **Random projections and metric embeddings**.

KEYWORDS

emulator, planar metric, spread reduction, Okamura-Seymour instance, slicing

ACM Reference Format:
Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. 2022. Almost-Linear ε-Emulators for Planar Graphs. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3519998

1 INTRODUCTION

Graph compression describes a paradigm of transforming a large graph G to a smaller graph G' that preserves, perhaps approximately, certain graph features such as distances or cut values. The algorithmic utility of graph compression is apparent — the compressed graph G' may be computed as a preprocessing step, reducing computational resources for subsequent processing and queries. This general paradigm covers famous examples like spanners, Gomory-Hu trees, and cut/flow/spectral edge-sparsifiers, in which case G' has the same vertex set as G, but fewer edges. Sometimes the compression is non-graphical and comprises of a small data structure instead of a graph G'; famous examples are distance oracles and distance labeling.

We study another well-known genre of compression, called *vertex sparsification*, whose goal is for G' to have a small vertex set. In this setting, the input graph G has a collection of k designated vertices T, called the *terminals*. The compressed graph G' should contain, besides the terminals in T, a small number of vertices and preserve a certain feature among the terminals. Specifically, we are interested in preserving the distances between terminals up to multiplicative factor $\alpha \geq 1$ in an edge-weighted graph (where the weights are interpreted as lengths). Formally, given a graph G with terminals $T \subseteq V(G)$, an *emulator* for G with *distortion* $\alpha \geq 1$ is a graph G' that contains the terminals, i.e., $T \subseteq V(G')$, satisfying

$$\forall x, y \in T, \quad \mathrm{dist}_G(x,y) \leq \mathrm{dist}_{G'}(x,y) \leq \alpha \cdot \mathrm{dist}_G(x,y), \quad (1)$$

where dist_G denotes the shortest-path distance in G (and similarly for G'). In the important case that $\alpha = 1+\varepsilon = e^{\Theta(\varepsilon)}$ for $0 \leq \varepsilon \leq 1$, we simply say that G' is an *ε-emulator*.[1] Notice that G' need not be a subgraph or a minor of G (in these two settings G' is known as a *spanner* and a *distance-approximating minor*).

We focus on the case where G is known to be planar, and thus require also G' to be planar (which excludes the trivial solution of a complete graph on T). This requirement is natural and also important for applications, where fast algorithms for planar graphs can be run on G' instead of on G. Such a requirement that G' has structural similarity to G is usually formalized by assuming that both G and G' belong to $\mathcal{F}$ for a fixed graph family $\mathcal{F}$ (e.g., all planar graphs). If $\mathcal{F}$ is a minor-closed family, one can further impose the stronger requirement that G' is a minor of G, and this clearly implies that G' is in $\mathcal{F}$.

Vertex sparsifiers commonly exhibit a tradeoff between accuracy and size, which in our case of an emulator G', are the distortion α and the number of vertices of G'. Let us briefly overview the known bounds for planar graphs. At one extreme of this tradeoff we have the "exact" case, where distortion is fixed to $\alpha = 1$ and we wish to bound the (worst-case) size of the emulator G' [8, 13, 24]. For planar graphs, the known size bounds are $O(k^4)$ [38] and $\Omega(k^2)$ [9, 41]. At the other extreme, we fix the emulator size to $|V(G')| = k$, i.e., zero non-terminals, and we wish to bound the (worst-case) distortion α [5, 6, 12, 21, 32]. For planar graphs, the known distortion bounds are $O(\log k)$ [20] and lower bound 2 [27].

*Full version of the paper can be found on arXiv. The full version includes an improvement on the size of the emulator, from $k^{1+o(1)}/\mathrm{poly}\,\varepsilon$ to $k/\mathrm{poly}\,\varepsilon$; a matching lower bound $\Omega(k/\varepsilon)$ for the emulator size for one-hole instance; as well as the first offline dynamic $(1+\varepsilon)$-approximate distance oracle for undirected planar graph with n vertices that has $O(\mathrm{poly}\log n)$ query and update time.

†Supported in part by ONR Award N00014-18-1-2364, Israel Science Foundation grant #1086/18, the Weizmann Data Science Research Center, and the Minerva Foundation.
‡Supported in part by NSF grant CCF-2006464.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3519998

[1] Our formal definition in Section 2 differs slightly (allowing two-sided errors), which affects our results only in some hidden constants.

Our primary interest is in minimizing the size-bound when the distortion α is $1+\varepsilon$, i.e., ε-emulators, a fascinating sweet spot of the tradeoff. The minimal loss in accuracy is a boon for applications, but it is usually challenging as one has to control the distortion over iterations or recursion. For planar graphs, the known size bounds for a distance-approximating minor are $\tilde{O}((k/\varepsilon)^2)$ [13] and $\Omega(k/\varepsilon)$ [38]. Improving the upper bound from quadratic to linear in k is an outstanding question that offers a bypass to the aforementioned $\Omega(k^2)$ lower bound for exact emulators ($\alpha = 1$). In fact, no subquadratic-size emulators for planar graphs are known to exist even when we allow the emulators to be arbitrary graphs, except for when the input is unweighted [8] or for trivial cases like trees.

Notation. Throughout the paper, we consider undirected graphs with non-negative edge weights. A *plane graph* refers to a planar graph together with a specific embedding in the plane. We suppress poly-logarithmic terms by writing $\tilde{O}(t) = t \cdot \operatorname{polylog} t$, and multiplicative factors that depend on ε by writing $O_\varepsilon(t) = O(f(\varepsilon) \cdot t)$. We write $\log^* t$ for the iterated logarithm of t.

1.1 Main Result

We design the first ε-emulators for planar graphs that have almost-linear size; furthermore, these emulators can be computed in near-linear time. These two efficiency parameters can be extremely useful, and we indeed present a few applications in Section 1.2.

Theorem 1.1. *For every n-vertex planar graph G with k terminals and parameter $0 < \varepsilon < 1$, there is a planar ε-emulator graph G' of size $|V(G')| = k^{1+o(1)}/\operatorname{poly}\varepsilon$. Furthermore, such an emulator can be computed deterministically in time $O\big(n \log^{O(1)} n/\operatorname{poly}\varepsilon\big)$.*

The result dramatically improves over the previous $\tilde{O}((k/\varepsilon)^2)$ upper bound of Cheung, Goranci and Henzinger [13]. Moreover, it breaks below the aforementioned lower bound $\Omega(k^2)$ for exact emulators ($\alpha = 1$) [9, 38, 41]. Unsurprisingly, our result is unlikely to extend to all graphs, because for some (bipartite) graphs, every minor with fixed distortion $\alpha < 2$ must have $\Omega(k^2)$ vertices [13]. See Table 1 for comparison to prior work.

Table 1: Distance emulators for planar graphs.

Distortion	Size (lower/upper)	Requirement	Reference
1	$\Omega(k^2)$	planar	[9, 41]
1	$O(k^4)$	minor	[38]
$1+\varepsilon$	$\Omega(k/\varepsilon)$	minor	[38]
$1+\varepsilon$	$\tilde{O}((k/\varepsilon)^2)$	minor	[13]
$1+\varepsilon$	$k^{1+o(1)}/\operatorname{poly}\varepsilon$	planar	Theorem 1.1
$O(\log k)$	k	minor	[20]

1.2 Algorithmic Applications

We present a few applications of our emulators to the design of fast $(1+\varepsilon)$-approximation algorithms for standard optimization problems on planar graphs. Our first application is to construct an approximate version of the multiple-source shortest paths data structure, called *ε-MSSP*: Preprocess a plane graph G and a set of terminals T on the outerface of G, so as to quickly answer distance queries between terminal pairs within $(1+\varepsilon)$-approximation. The preprocessing time of our data structure is $O_\varepsilon(n \operatorname{poly}(\log^* n))$, which for fixed $\varepsilon > 0$ is faster than Klein's $O(n \log n)$-time algorithm [35] for the exact setting when $\varepsilon = 0$. Both algorithms have the same query time $O(\log n)$.

Theorem 1.2. *Given a parameter $0 < \varepsilon < 1$, an n-vertex plane graph G and a set of terminals T all lying on the boundary of G, one can preprocess an ε-MSSP data structure on G with respect to T in time $O(n \operatorname{poly}(\log^* n)/\operatorname{poly}\varepsilon)$, that answers queries in time $O(\log n)$.*

Our second application is an $O_\varepsilon(n \log^* n)$-time algorithm to compute $(1+\varepsilon)$-approximate minimum (s,t)-cut in planar graphs, which for fixed $\varepsilon > 0$ is faster than the $O(n \log\log n)$-time exact algorithm by Italiano, Nussbaum, Sankowski, and Wulff-Nilsen [31].

Theorem 1.3. *Given an n-vertex planar graph G with two distinguished vertices $s, t \in V(G)$ and a parameter $0 < \varepsilon < 1$, one can compute a $(1+\varepsilon)$-approximation to the minimum (s,t)-cut in G in time $O(n \log^* n/\operatorname{poly}\varepsilon)$.*

Our third application is an $O_\varepsilon(n \log n \log^* n)$-time algorithm to compute a $(1+\varepsilon)$-approximate diameter in planar graphs, which for fixed $0 < \varepsilon < 1$ is faster than the $O(n \log^2 n + \varepsilon^{-5} n \log n)$-time algorithm of Chan and Skrepetos [7] (which itself improves over Weimann and Yuster [49]).

Theorem 1.4. *Given an n-vertex planar graph G and a parameter $0 < \varepsilon < 1$, one can compute a $(1+\varepsilon)$-approximation to its diameter in time $O(n \log n \log^* n + n \log n/\operatorname{poly}\varepsilon)$.*

As the applications can be derived from applying the ε-emulator construction with bootstrapping, we only present the bootstrapping idea and postpone the complete proofs to the full version.

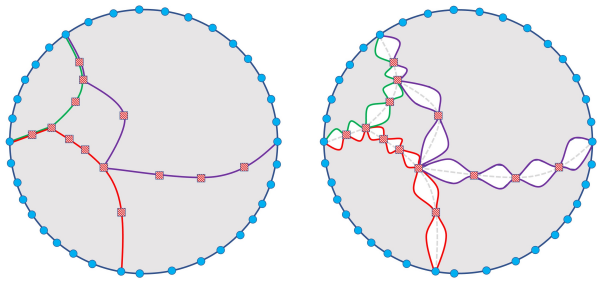
1.3 Technical Contributions

A central technical contribution of this paper is to carry out a *spread reduction* for the all-terminal-pairs shortest path problem when the input graph can be embedded in the plane and the terminals all lie on the outerface; the *spread* is defined to be the ratio between the largest and the smallest distances between terminals. Spread reduction is a crucial preprocessing step for many optimization problems, particularly in Euclidean spaces or on planar graphs [4, 15, 22, 32, 47], that replaces an instance with a large spread with one or multiple instances with a bounded spread. In many cases, one can reduce the spread to be at most polynomial in the input size. However, we are not aware of previous work that achieves such a reduction in our context, where many pairs of distances have to be preserved all at once. In fact, even after considerable work we only managed to reduce the spread to be sub-exponential.

We now provide a bird-eye's view of our emulator construction. The emulator problem on plane graphs with an arbitrary set of terminals can be reduced to the same problem on plane graphs, but with the strong restriction that all the terminals lies on a constant number of faces, known as *holes* (cf. Section 5), using a separator decomposition that splits the number of vertices and terminals evenly; such a decomposition (called the *r-division*) can be computed efficiently [23, 36]. From there we can further slice the graph open

into another plane graph with all the terminals on a single face, which without loss of generality we assume to be the outerface. We refer to it as a *one-hole instance*.

To construct an emulator for a one-hole instance G we adapt a recursive *split-and-combine* strategy (cf. Section 3). We will attempt to split the input instance into multiple one-hole instances along some shortest paths that distribute the terminals evenly (cf. Lemma 3.2). Every time we slice the graph G open along a shortest path P, we compute a small collection of vertices on P called the *portals*, that approximately preserve the distances from terminals in G to the vertices on P. The portals are duplicated during the slicing along P and added to the terminal set (i.e., become terminals) at each piece incident to P, to ensure that further processing will (approximately) preserve their distances as well. We emphasize that the naive idea of placing portals at equally-spaced points along P is not sufficient, as some terminals in G might be arbitrarily close to P. Instead, we place portals at exponentially-increasing intervals from both ends of P. After splitting the original instance into small enough pieces by recursively slicing along shortest paths and computing the portals, we compute exact emulators for each piece using any of the polynomial-size construction [9, 38]. Next we glue these small emulators back along the paths by identifying multiple copies of the same portal into one vertex. See Figure 1.



(a) (b)

Figure 1: Illustration of the split-and-combine process for a one-hole instance.

Let U be the set of terminals in the current piece, and let $r := |U|$. We need the portals to be dense enough so that only a small error term, of the form $r^{-\delta}$ (meaning that the distortion increases multiplicatively by $1 + r^{-\delta}$) will be added to the distortion of the emulator after the gluing, as this will eventually guarantee (through more details like the stopping condition of the recursion) that the final distortion is $1 + \varepsilon$ and the final emulator size has polynomial dependency on ε^{-1}. At the same time, the number of portals cannot be too large, as they are added to the terminal set, causing the number of terminals per piece to go down slowly and creating too many pieces, and in the end the size of the combined emulator might be too big. It turns out that the sweet spot is to take roughly $L_r := r/\log^2 r$ portals. Calculations show that in such case the portals preserve distances up to error term $\log \Phi / L_r$, where Φ is the *spread* of the terminal distances (cf. Claim 4.4). When $\Phi \le 2^{r^{0.9}}$, we will get the polynomially-small $\tilde{O}(r^{-0.1})$ error term needed for the gluing (cf. Section 4.2). However, even when the original input has a polynomial spread to start with, in general we cannot control the spread of all the pieces occurring during the split-and-combine process, because portals are added to the terminal sets. Therefore a new idea is needed.

When $\Phi > 2^{r^{0.9}}$, we need to tackle the spread directly. We perform a *hierarchical clustering* of the terminals (cf. Section 4.3). At each level i, we connect two clusters of terminals from the previous level $i-1$ using an edge if their distance is at most r^{2i}; then we group each connected component into a single cluster. The key to the spread reduction is the idea of *expanding clusters*. A cluster S is *expanding* if its parent cluster $\hat{S}$ is at least $\sim e^{r^{-0.7}}$-factor bigger. Intuitively, if all clusters are expanding, then the number of levels in the hierarchical clustering must be at most $r^{0.7}$, and therefore the spread must be at most sub-exponential. So in the high-spread case some non-expanding cluster must exist.

If such non-expanding cluster S is of moderate size (that is, in between $r/5$ and $4r/5$) (cf. Section 4.3.1), we construct a collection of *non-crossing* shortest paths between terminals in S (non-crossing means that no two paths with endpoint pairs (s_1, s_2) and (t_1, t_2) have their endpoints in an interleaving order (s_1, t_1, s_2, t_2) on the outerface) in which no two paths intersect except at their endpoints. Again compute portals on the paths from every terminal in $\hat{S} \setminus S$, but now using ε_r-covers [48] for $\varepsilon_r := r^{-0.1}$, and split along the paths to create sub-instances. Because the cluster is non-expanding and has moderate size, the number of terminals in $\hat{S} \setminus S$ is at most $(e^{r^{-0.7}} - 1)|S| \le r^{0.3}$, and thus the number of portals is $O(r^{0.3}/\varepsilon_r) \le O(r^{0.4})$, which is a gentle enough increase in the number of terminals. The hard part is to argue that the portals created are sufficient for the recombined instance to be an emulator. This can be done by observing that terminal pairs among $U \setminus \hat{S}$ are far apart, and similarly when one terminal is from S and the other is from $U \setminus \hat{S}$; hence only terminal pairs involving $\hat{S} \setminus S$ have to be dealt with using properties of ε_r-covers (cf. Claim 4.8).

If there are no non-expanding clusters with moderate size (cf. Section 4.3.2), we find a non-expanding cluster $\tilde{S}$ of lowest level that contains most of the terminals, and construct a collection of non-crossing shortest paths between terminals in $\tilde{S}$ like the previous case. However this time, after computing the $r^{-0.1}$-covers and splitting along the paths, there might be one instance containing too many terminals. In this case, we find *every* non-expanding cluster S of *maximal level*; such clusters must all lie within $\tilde{O}(r^{0.7})$ levels from $\tilde{S}$ because we cannot have nested expanding clusters for $\tilde{O}(r^{0.7})$ consecutive levels. The Monge property guarantees that the shortest paths generated by the union of these maximal-level non-expanding clusters must be non-crossing because all such clusters are disjoint (cf. Observation 4.7). Now if we split the graph based on the path set generated, each resulting instance either has moderate size, or must have small spread, and we safely fall back to the earlier cases.

Applications. A widely adopted pipeline in designing efficient algorithms for distance-related optimization problems on planar graphs in recent years consists of the following steps:

(1) Decompose the input planar graph into small pieces each of size at most r with a small number of boundary vertices and $O(1)$ holes, called an *r-division* (see Frederickson [23] and Klein-Mozes-Sommer [36];

(2) Process each piece so that all-pairs shortest paths between boundary vertices within a piece can be extracted efficiently by the *multiple-source shortest paths* algorithm for planar graphs (Klein [35]);
(3) Further process each piece into a *compact data structure* that supports efficient min-weight-edge queries and updates (SMAWK[1], Fakcharoenphol and Rao [19]);
(4) Compute shortest paths in the original graph in a problem-specific fashion, now with each piece replaced with the compact data structure, using a *modified Dijkstra algorithm* (Fakcharoenphol and Rao [19]).

The conceptual role of our planar emulators is an alternative to Step 3. In a sense, the reason for the development of the aforementioned machinery and complex algorithms is to get around the size lower bound in representing the all-pairs distances for the pieces. The benefit of replacing the data structure with a single planar emulator is that the whole graph stays planar. One can then simply replace Step 4 with the standard Dijkstra algorithm (or even better, with the $O(n)$-time algorithm for planar graphs by Henzinger *et al.* [29]). More importantly, one can *recurse* on the resulting graph when appropriate, and compress the graph further and further with small additive errors slowly accumulated. This allows us to construct an ε-emulator that is sub-linear in the size of each piece in linear time (up to $O_\varepsilon(\log^* n)$ factors).

1.4 Related Work

In addition to emulators, there are other lines of research on graph compression that preserve distance information. Among them the most studied objects are spanners and preservers (in which the sparsifier is required to be a subgraph of the input graph) and distance oracles (that is a data structure that report exact or approximate distances between pairs of vertices). We refer the reader to the excellent survey [2].

Another type of vertex sparsifiers that are extensively studied are cut/flow sparsifiers. There are rich lines of works for constructing vertex sparsifiers that preserve cut/flow values exactly [11, 24, 28, 33, 34, 39, 40] or approximately [3, 10, 14, 16, 25, 26, 42, 43].

2 PRELIMINARIES

All logarithms are to the base of 2. All graphs are simple and undirected. Let G be a connected graph. A vertex $v \in V(G)$ is called a *cut vertex* of G if the graph $G \setminus \{v\}$ is disconnected. The cut vertices of a plane graph G can be computed in time $O(|V(G)| + |E(G)|)$. Let G be a graph with an edge-weight function $w\colon E(G) \to \mathbb{R}_+$. The weight of a path P is defined as $w(P) := \sum_{e\in E(P)} w(e)$. The shortest-path distance between two vertices u and v is denoted by $\mathrm{dist}_G(u,v)$. For a subset S of vertices in G, we define $\mathrm{diam}_G(S) := \max_{u,u'\in S} \mathrm{dist}_G(u,u')$. For a pair of disjoint subsets of vertices (S,S') in G, we define $\mathrm{dist}_G(S,S') := \min_{u\in S,u'\in S'} \mathrm{dist}_G(u,u')$.

Emulators. Throughout, we consider graph G equipped with a special set of vertices T, called *terminals*. We refer to the pair (G,T) as an *instance*. Let (G,T) and (H,T) be a pair of instances with the same set of terminals, and let $\varepsilon \in [0,1]$. We say that H is an *ε-emulator* for G with respect to T, or equivalently, instance (H,T) is an ε-emulator for instance (G,T) if

$$\forall x,y \in T, \quad e^{-\varepsilon}\cdot \mathrm{dist}_G(x,y) \le \mathrm{dist}_H(x,y) \le e^{\varepsilon}\cdot \mathrm{dist}_G(x,y). \quad (2)$$

Throughout, we use Equation (2) as the definition of an ε-emulator instead of Equation (1); but since we restrict our attention to $\varepsilon < 1$, the two definitions are equivalent up to scaling ε by a constant factor. By definition, if (H,T) is an ε-emulator for (G,T), then (G,T) is an ε-emulator for (H,T). Moreover, if (G,T) is an ε-emulator for (G',T) and (G',T) is an ε'-emulator for (G'',T), then (G,T) is an $(\varepsilon+\varepsilon')$-emulator for (G'',T).

In this paper we mostly consider instance (G,T) where graph G is a plane graph, which we refer to as *planar instances*. We say that a planar instance (G,T) is an *h-hole instance* for an integer $h > 0$ if the terminals lie on at most h faces in the embedding of G. The faces incident to some terminals are called *holes*. Notice that in a one-hole instance (G,T), we can safely assume all the terminals in T lie on the outerface G. By definition, a 0-emulator preserves distances exactly, i.e., $\mathrm{dist}_G(x,y) = \mathrm{dist}_{G'}(x,y)$ for all $x,y \in T$.

Theorem 2.1 (Chang-Ophelders [9, Theorem 1]). *Given one-hole instance (G,T) with $n := |V(G)|$ and $k := |T|$, one can compute a 0-emulator (G',T) for (G,T) of size $|V(G')| \le k^2$. The running time of the algorithm is $O((n+k^2)\log n)$.*

Crossing pairs and the Monge property. Let (G,T) be a one-hole instance. Assume that no terminal in T is a cut vertex of G, every terminal appears exactly once as we traverse the boundary of the outerface. Let $(t_1,t_2),(t_1',t_2')$ be two terminal pairs whose four terminals are all distinct. We say that the pairs $(t_1,t_2),(t_1',t_2')$ are *crossing* if the clockwise order in which these terminals appear on the boundary is either (t_1,t_1',t_2,t_2') or (t_1,t_2',t_2,t_1'); otherwise we say that they are *non-crossing*. A collection $\mathcal{M}$ of pairs of terminals in T is called *non-crossing* if every two pairs in $\mathcal{M}$ is non-crossing. Sometimes we abuse the language and say that a set of shortest paths $\mathcal{P}$ in G is *non-crossing* when the collection of endpoint pairs for the paths is non-crossing. The *Monge property*[2] states that, for every one-hole instance (G,T) and every crossing pairs of terminals (t_1,t_2) and (t_1',t_2'),

$$\mathrm{dist}_G(t_1,t_2) + \mathrm{dist}_G(t_1',t_2') \ge \mathrm{dist}_G(t_1',t_2) + \mathrm{dist}_G(t_1,t_2').$$

Well-structured sets of shortest paths. Consider a graph G and a collection $\mathcal{P}$ of shortest paths in G. We say that the set $\mathcal{P}$ is *well-structured* if for every pair of paths (P,P') in $\mathcal{P}$, $P \cap P'$ is a single subpath of both P and P'. It is not hard to see that every collection of shortest paths in G is well-structured if the shortest path between any two vertices in G is unique. Such condition can be enforced with high probability if we perturb the edge-weights in G slightly and apply the *isolation lemma* [17, 45]. A deterministic lexicographic perturbation scheme that guarantees the uniqueness of shortest paths in an n-vertex plane graph can be computed in $O(n)$ time [17]. The following lemma is proved in the full version of the paper.

Lemma 2.2. *Given a one-hole instance (G,T) and a non-crossing collection $\mathcal{M}$ of pairs of terminals in T, one can compute a well-structured set $\mathcal{P}$ of shortest paths, one for each pair of terminals in T in $O(|E(G)|\cdot\log|\mathcal{M}|)$ time.*

[2] Technically, this is known as the *cyclic Monge property* [9].

Therefore from here on we assume that all the planar graphs we consider have unique shortest path between every pair of vertices, and every collection of shortest paths is well-structured.

ε-cover. We use the notion of *ε-cover* [37, 48]. Let $\varepsilon \in (0,1)$ be a parameter. Let G be a graph and let P be a shortest path in G connecting some pair of vertices. Consider now a vertex v in G that does not belong to path P. An *ε-cover* of v on P is a subset S of vertices in P such that, for each vertex $x \in V(P)$, taking the detour from v to some $y \in S$ then to x is a $(1+\varepsilon)$-approximation to the shortest path from v to x, i.e., there exists $y \in S$ for which $\mathrm{dist}_G(v,y) + \mathrm{dist}_G(y,x) \leq (1+\varepsilon) \cdot \mathrm{dist}_G(v,x)$. Small ε-cover of size $O(1/\varepsilon)$ is known to exist.

Theorem 2.3 (Thorup [48, Lemma 3.4]). *Let $\varepsilon \in (0,1)$ be a constant. For every shortest path P in some graph G and every vertex $v \notin P$, there is an ε-cover of v on P with size $O(1/\varepsilon)$. Moreover, such an ε-cover can be computed in $O(|E(G)|)$ time.*

We emphasize that choosing $O(1/\varepsilon)$ "portals" at equal distance on the path P as in Klein-Subramanian [37, Lemma 4] is not sufficient, because the distance from v to P might be much smaller than the length of P. The linear-time construction is not stated in Lemma 3.4 of [48], but it can be inferred from their proof. In fact, we will use the following construction that allows us to efficiently compute the union of ε-covers of a subset Y of vertices along the boundary of plane graph; the proof is a simple divide-and-conquer similar to Reif [46], which we omit here.

Lemma 2.4. *Let $\varepsilon \in (0,1)$ be a constant and G is a plane graph. Given a subset Y of vertices that lie on the same face of G and a shortest path P connecting a pair of vertices in G, we can compute the union of ε-covers of each vertex in Y on P in $O(|E(G)| \cdot \log |Y|)$ time.*

3 EMULATORS FOR ONE-HOLE INSTANCES

In this section and the next one we design a near-linear time algorithm for constructing ε-emulators for one-hole instances, as stated in Theorem 3.1. We say that an ε-emulator (G', T) for a one-hole instance (G, T) is *aligned* if (G', T) is also a one-hole instance, and the circular orderings of the terminals on the outerfaces of G and of G' are identical.

Theorem 3.1. *Given a parameter $\varepsilon \in (0,1)$ and one-hole instance (G,T) with $k := |T|$, one can compute an aligned ε-emulator for (G,T) of size $k \log^{O(1)} k/\mathrm{poly}\,\varepsilon$ in $O\big((n+k^2)\log^{O(1)} n/\mathrm{poly}\,\varepsilon\big)$ time.*

In the rest of this section and the next one, all the emulators are aligned, and therefore we omit the word "aligned" from now on. We describe the algorithm and proof for Theorem 3.1 in Section 3.1, with the help of the core decomposition lemma (cf. Lemma 3.2). The proof to Lemma 3.2 itself is deferred to Section 4.

3.1 The Algorithm and Its Analysis

In this subsection we describe the algorithm for Theorem 3.1 and provide its analysis. Let (G,T) be the input one-hole instance. The algorithm consists of two stages. The first stage iteratively decomposes (G,T) into smaller one-hole instances, and the second stage computes emulators for these small instances and then combines them into an emulator for (G,T).

Algorithm. Throughout the algorithm we maintain a collection $\mathcal{H}$ of one-hole instances, that is initialized to be $\mathcal{H} = \{(G,T)\}$. Set $\lambda := c^* \log^2 k/\varepsilon^{20}$, where $k := |T|$ and $c^* > 0$ is a large enough constant. In the first stage, we repeatedly replace a one-hole instance $(H,U) \in \mathcal{H}$ where $|U| > \lambda$ with new one-hole instances obtained by applying the algorithm from Lemma 3.2 to (H,U), until all one-hole instances in $\mathcal{H}$ have $|U| \leq \lambda$. The core of our construction is the following lemma.

Lemma 3.2. *Given one-hole instance (H,U) with $r := |U|$ and threshold $\lambda := c^* \log^2 k/\varepsilon^{20}$, one can compute a collection of one-hole instances $\{(H_1,U_1),\ldots,(H_s,U_s)\}$, such that*

- $U \subseteq \big(\bigcup_{1\leq i\leq s} U_i\big)$;
- $|U_i| \leq 9r/10$ *for each* $1 \leq i \leq s$;
- $\sum_{i:|U_i|\leq\lambda} |U_i| \leq O(r\log^2 r)$; *and*
- $\sum_{i:|U_i|>\lambda} |U_i| \leq r \cdot \big(1 + O(\frac{1}{\log^2 r})\big)$.

Moreover, given an ε-emulator (H_i', U_i) for each (H_i, U_i), algorithm Combine *computes for (H,U) an $\big(\varepsilon + O(\frac{\log^4 r}{r^{0.1}})\big)$-emulator (H', U):*

$$|V(H')| \leq \left(\sum_{1\leq i\leq s} |V(H_i')|\right) + O(r\log^2 r).$$

The running time of both algorithms is at most $O\big((|V(H)| + r^2)\cdot \log r \cdot \log |V(H)|\big)$.

We prove this lemma in Section 4, and in the remainder of this subsection we use it to complete the proof of Theorem 3.1.

We associate with the decomposition process a *partitioning tree τ*. Its nodes are all the one-hole instances that ever appear in the collection $\mathcal{H}$. Its root node is the initial one-hole instance (G,T), and every tree node (H,U) has children nodes corresponding to the new instances $(H_1,U_1),\ldots,(H_s,U_s)$ generated by Lemma 3.2. The leaves of τ are those that are in $\mathcal{H}$ at the end of the first stage. (To avoid ambiguity, we refer to elements in $V(\tau)$ as *nodes* and elements in $V(H)$ as *vertices*.)

We now describe the second stage of the algorithm. For each one-hole instance (H,U) in $\mathcal{H}$ at the end of the first stage, compute a 0-emulator (H',U) for (H,U) using the algorithm from Theorem 2.1.[3] We then iteratively process the non-leaf nodes in τ inductively in a bottom-up fashion: Given a non-leaf node (H,U) with children $(H_1,U_1),\ldots,(H_s,U_s)$, let (H_i',U_i) be the emulator computed for (H_i,U_i) by induction. Apply algorithm Combine from Lemma 3.2 to the emulators $(H_1',U_1),\ldots,(H_s',U_s)$ to obtain an emulator (H',U) for instance (H,U). After all nodes in τ have been processed, output the emulator (G',T) constructed for the root node (G,T).

We proceed to show that the instance (G',T) computed by the above algorithm satisfies all the properties required in Theorem 3.1.

Size analysis. We need to bound the size of the emulator G' by $k\log^{O(1)} k/\mathrm{poly}\,\varepsilon$. In what follows, $\mathcal{H}$ denotes this set at the end of the first stage.

- For each node (H,U) in the partitioning tree τ, we define $f(H,U) := \hat{c}\cdot|U|\log^2|U|$, where $\hat{c} > 0$ is a constant independent to (H,U) that is greater than all constants hidden

[3] This step can use any 0-emulator that has size poly k and can be constructed in time $\tilde{O}(n + \mathrm{poly}\,k)$, and we conveniently use Theorem 2.1.

in the big-O notations in the statement of Lemma 3.2. By Theorem 2.1 and Lemma 3.2, the final emulator has size

$$|V(G')| \le \sum_{(H,U)\in\mathcal{H}} |U|^2 + \sum_{(H,U)\in V(\tau)} f(H,U).$$

- We bound the term $\sum_{(H,U)\in V(\tau)} f(H,U)$ via a charging scheme as follows. Consider a node (H,U) in τ. We charge the value of $f(H,U)$ uniformly to all terminals in U, so each terminal has a charge of at most $\hat{c}\log^2|U| \le \hat{c}\log^2 k$. The height of τ is at most $O(\log k)$, because at every node that is split by Lemma 3.2, the number of terminals decreases by at least a factor of $9/10$. So every terminal in $\bigcup_{(H,U)\in\mathcal{H}} U$ is charged at most $O(\log k)$ times, thus has a total charge at most $O(\log^3 k)$.
- It suffices to bound the total number of terminals in all resulting one-hole instances in $\mathcal{H}$ by $k\log^{O(1)} k$, because then the total size of emulators (H',U) for all resulting one-hole instances in $\mathcal{H}$ is

$$\sum_{(H,U)\in\mathcal{H}} |U|^2 \le \max_{(H,U)\in\mathcal{H}}\{|U|\}\cdot \sum_{(H,U)\in\mathcal{H}} |U| \le \lambda\cdot k\log^{O(1)} k = k\log^{O(1)} k/\mathrm{poly}\,\varepsilon.$$

  Therefore, $|V(G')|$ is at most

$$\sum_{(H,U)\in\mathcal{H}} \left(|U|^2 + |U|\cdot O(\log^3 k)\right) = k\cdot\log^{O(1)} k/\mathrm{poly}\,\varepsilon.$$

It remains to bound the total number of terminals in all one-hole instances in $\mathcal{H}$, which we do next via a charging scheme. Let (H,U) be a node in τ with children $(H_1,U_1),\ldots,(H_s,U_s)$.

- For instances (H_i,U_i) with $|U_i|\le\lambda$ (which will all be in $\mathcal{H}$ at the end of first stage), charge every vertex in U_i to vertices in U. Since $\sum_{i:|U_i|\le\lambda}|U_i| \le O(\log^2|U|\cdot|U|) \le O(\log^2 k\cdot|U|)$, each vertex of U gets a charge of $O(\log^2 k)$ this way. We call these charge *inactive*.
- For instances (H_i,U_i) with $|U_i|>\lambda$, let U' be the set of all new vertices, i.e., they appear in some set U_i but not in U; we have $|U'| \le O(|U|/\log^2|U|)$ by Lemma 3.2. Charge every vertex in U' uniformly to vertices in U, so each vertex gets $O(1/\log^2|U|)$ charge. We call these charge *active*.

The total inactive charge on each vertex of T is $O(\log^3 k)$ because τ has height $O(\log k)$. As for the total active charge to each vertex in T, a quick calculation shows that it is at most $O(1/(\log_{(10/9)}\lambda - 1)) \le 1/2$. Note that this only accounts for the *direct active charge*. For example, some terminal does not belongs to the initial one-hole instance (G,T), that was first actively charged to the terminals in T, can in turn be actively charged by some other terminals later. We call such charge *indirect active charge*. The total direct and indirect active charge for each terminal in T is at most $1/2+(1/2)^2+\cdots\le 1$.

Altogether, each terminal in T is charged $O(\log^3 k)$. Therefore, we conclude that the total number of terminals in all resulting instances in $\mathcal{H}$ is bounded by $O(k\log^3 k)$, which, from the above discussion, implies that the total size of emulators (H',U) for all resulting one-hole instances in $\mathcal{H}$ is $k\log^{O(1)} k/\mathrm{poly}\,\varepsilon$.

Correctness. It remains to show that (G',T) is an ε-emulator for (G,T). Recall that we have associated with the algorithm in first stage a (rooted) partitioning tree τ. We now define, for each tree node (H,U), a value $\varepsilon_{(H,U)}$ as follows. If (H,U) is a leaf node, we define $\varepsilon_{(H,U)} := 0$. Otherwise, (H,U) is a non-leaf node with child nodes in τ be $(H_1,U_1),\ldots,(H_s,U_s)$. Denote $r := |U|$, and let $c>0$ be a large enough constant that is greater than the constants hidden in all big-O notations in Lemma 3.2 and $c < (c^*)^{1/20}$. We define

$$\varepsilon_{(H,U)} := \frac{c\log^4 r}{r^{0.1}} + \max\{\varepsilon_{(H_1,U_1)},\ldots,\varepsilon_{(H_s,U_s)}\}.$$

From the properties of the algorithm Combine, it is easy to verify that for each node (H,U) in τ, the one-hole instance (H',U) we construct is an $\varepsilon_{(H,U)}$-emulator for (H,U).

We now show that $\varepsilon_{(G,T)} \le \varepsilon$. Observe that there are integers $r_1,\ldots,r_t$ with $r_1\le k$, $r_t\ge\lambda$, such that for each $1\le i\le t-1$, $r_i \ge (10/9)\cdot r_{i+1}$, $\varepsilon_{(G,T)} = \sum_{1\le i\le t} c\log^4 r_i/r_i^{0.1}$. A quick calculation gives us $\varepsilon_{(G,T)} \le c\cdot(\log\lambda)^4/\lambda^{0.1}$. Since c is a constant, and recall that we have set $\lambda = c^*\log^2 k/\varepsilon^{20}$ where $c^* > c^{20}$ is large enough, so $\varepsilon_{(G,T)} < \varepsilon$, and therefore (G',T) is an ε-emulator for (G,T).

Running time. Every time we implement the algorithm from Lemma 3.2 to split some instance in $(H,U)\in\mathcal{H}$ with $n := |H|$ and $r := |U|$, we charge its running time (and also the time for Combine): (i) the $n\log^{O(1)} r\cdot\log n$ term uniformly to all vertices in H, so every vertex is charged at most $\log^{O(1)} r\cdot\log n$; and (ii) the $r^2\log^{O(1)} r\cdot\log n$ term uniformly to all terminals in U, so every terminal in U is charged at most $r\log^{O(1)} r\cdot\log n$. Using similar charging scheme as described above, we get that the total running time is at most $\log^{O(1)} k\cdot\log n$ times the total number of vertices in all resulting one-hole instances in $\mathcal{H}$, which is at most $n+(k\log^{O(1)} k/\mathrm{poly}\,\varepsilon)$, plus $k\log^{O(1)} k\cdot\log n$ times the total number of terminals in all resulting one-hole instances in $\mathcal{H}$, which is at most $k\log^{O(1)} k)/\mathrm{poly}\,\varepsilon$. Therefore, the total running time of the algorithm is $(n+k^2)\log^{O(1)} k\cdot\log n/\mathrm{poly}\,\varepsilon$.

4 RECURSIVE EMULATOR CONSTRUCTION

In this subsection we provide the proof of Lemma 3.2. We first introduce the basic graph operations Split and Glue in Section 4.1. Then we describe the algorithm and its analysis. The algorithm described in this section can be implemented in $O\big((n+r^2)\cdot\log r\cdot\log n\big)$ time; see the full version for details.

4.1 Splitting and Gluing

In this subsection we introduce building blocks for the divide-and-conquer: procedures Split and Glue. We will decompose a single one-hole instance (H,U) into many small one-hole instances using procedure Split, compute emulators for each of them, and then glue the collection of small emulators together into an emulator for (H,U) using procedure Glue.

Splitting. The input to procedure Split consists of

- a one-hole instance (H,U);
- a non-crossing set $\mathcal{P}$ of shortest paths in H connecting pairs of terminals in U; and
- a subset Y of vertices on the union of shortest paths in $\mathcal{P}$; Y must contain all endpoints of paths in $\mathcal{P}$.

The output of procedure Split is a collection of one-hole instances constructed as follows.

Consider the plane graph H with all the terminals in U lying on the outerface of H. Because all the paths in $\mathcal{P}$ are shortest paths in H, each path must be simple and connecting two terminals in U. Given any plane graph H and simple path P, we can *slice*[4] H open along the path P by duplicating every vertex and edge of P to create another path P' identical to P, together bounding a common new boundary component. The set of edges incident to each vertex on P are split into two sides naturally based on the cyclic order around the vertex.

The output of procedure Split is simply the collection $\mathcal{H}$ of one-hole instances formed by iteratively slicing (the remaining of) H using shortest paths from $\mathcal{P}$. Note that each vertex $y \in Y$ may now belong to multiple instances in $\mathcal{H}$. We call them *copies* of y.

Gluing. We now describe procedure Glue. Assume that we have applied the procedure Split to the one-hole instance (H, U), and $\mathcal{H} := \{(H_R, U_R)\}$ is the collection of one-hole instances produced. The input to procedure Glue consists of

- one emulator Z_R for each one-hole instance (H_R, U_R) in $\mathcal{H}$;
- the same vertex subset Y given as the input to Split.

The output of procedure Glue is an emulator for (H, U), which is constructed as follows.

We start by taking the union of all emulators $\{Z_R\}$, and identifying, for each vertex $y \in Y$, all copies of the vertex. We denote by $\hat{Z}$ the resulting graph. We then add to $\hat{Z}$ all the *branch vertices* of $\mathcal{P}$ vertices that are not in Y (and thus have not been merged), where the *branch vertices* of $\mathcal{P}$ in H are those vertices that has degree at least three in the union of shortest paths in $\mathcal{P}$. For each emulator Z_R, denote the cyclic sequence of all terminals in U_R and the copies of branch vertices appearing around the outerface of Z_R in counter-clockwise order as $\langle v_1, \ldots, v_m \rangle$. For each branch vertex v_i not in U_R, we add an edge connecting v_i to the previous vertex v_{i-1} (might be a terminal or another branch vertex), with the edge-weight $\mathrm{dist}_H(v_i, v_{i-1})$; similarly, add an edge connecting v_i to v_{i+1} with edge-weight $\mathrm{dist}_H(v_i, v_{i+1})$. (See Figure 2 for an illustration.) We denote the resulting graph Z after processing all the emulators Z_R inside $\hat{Z}$. Graph Z is naturally a plane graph by inheriting the embeddings of all Z_Rs. By the assumption that Y contains all the endpoints of paths in $\mathcal{P}$, every vertex in U shows up uniquely on the outerface of Z. The size of Z is bounded by $\sum_R |Z_R|$ plus the number of branch vertices.

It is easy to verify that both procedures Split and Glue can be implemented in $O(|V(H)|)$ time. Now we summarize the behavior of the algorithms with the following three claims; let $((H, U), \mathcal{P}, Y)$ be a valid input to procedure Split and $\mathcal{H}$ be the corresponding output throughout the description. The proofs of Claim 4.1, 4.2, and 4.3 are deferred to the full version.

Claim 4.1. *Graph H has at most $O(|U| \cdot \log^2 |U|)$ branch vertices.*

Claim 4.2. *The total number of terminals in all one-hole instances in $\mathcal{H}$ after procedure Split is*

$$\sum_{(H_R, U_R) \in \mathcal{H}} |U_R| \le O\big(|U| \cdot \log^2 |U| + |Y|\big),$$

[4]The slicing operation, which can be traced back to Reif [46] (when describing the minimum-cut algorithm by Itai-Shiloach [30]), is sometimes referred to as *cutting* [18] or *incision* [44] in the literature.

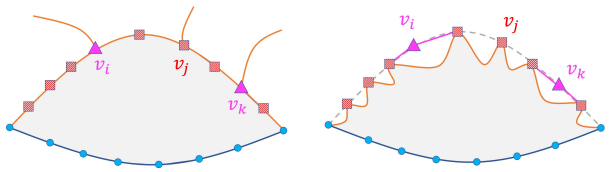


Figure 2: Adding edges that connect to branch vertices. *Left:* Graph H_R: v_i, v_j, v_k are branch vertices, and only vertices v_i, v_k belong to set $V^* \setminus U_R$. *Right:* Graph Z_R in Z: v_i, v_k do not belong to Z_R but they are connect to vertices of U_R by edges (purple).

and for any $2 < \mu < |U|/2$,

$$\sum_{(H_R, U_R) \in \mathcal{H}: |U_R| \ge \mu} |U_R| \le \big(|U| + O(|Y \setminus U|)\big) \cdot \left(1 + \frac{1}{\mu - 2}\right).$$

Claim 4.3. *Let $(\hat{H}, U)$ be the instance obtained by applying the procedure Glue directly to the instances in $\mathcal{H}$. The output (Z, U) from procedure Glue when applying to the emulators of instances in $\mathcal{H}$ is an ε-emulator for $(\hat{H}, U)$.*

Later in the proof we will use the divide-and-conquer technique twice. Because the set Y we choose to glue along is different each time, we will postpone the statements that relate $(\hat{H}, U)$ to the original instance (H, U) to the later sections in context. (See Claim 4.4 and Claim 4.8).

Now we focus on proving Lemma 3.2 for any instance (H, U) where no vertex in U is a cut vertex of graph H; in other words, if we traverse the boundary of the face that contains all terminals in U, every terminal of U appear exactly once. The reduction from general instances to this special case can be found in the full version.

Spread of an instance. Let (H, U) be a planar instance. The *spread* of (H, U) is defined to be

$$\Phi(H, U) := \frac{\max_{u, u' \in U} \mathrm{dist}_H(u, u')}{\min_{u, u' \in U} \mathrm{dist}_H(u, u')}.$$

Denote the spread of instance (H, U) by $\Phi := \Phi(H, U)$. We distinguish between the following two cases, depending on whether or not Φ is small or large.

4.2 Small Spread Case

In this case we assume $\Phi \le 2^{r^{0.9} \log^2 r}$. We will employ the procedure Split in order to decompose the one-hole instance (H, U) into smaller instances. Throughout this case, we use parameters

$$L_r := r/100 \log^2 r \quad \text{and} \quad \varepsilon_r := \log \Phi / L_r,$$

so $\varepsilon_r = O(\log^4 r / r^{0.1})$.

Balanced terminal pairs. Let (H, U) be a one-hole instance with $U = \{u_1, \ldots, u_r\}$, where the terminals are indexed according to the order in which they appear on the outerface. We say that a pair of terminals (u_i, u_j) (with $i < j$) is a *c-balanced pair* for some parameter $1/2 < c < 1$, if and only if $j - i \le c \cdot r$ and $i + r - j \le c \cdot r$. In other words, the terminals u_i and u_j separate the outer boundary

into two segments, each contains at most c-fraction (and therefore at least $(1-c)$-fraction) of the terminals.

We first compute the $(3/4)$-balanced pair u, u' of terminals that, among all $(3/4)$-balanced pairs of terminals in U, minimizes the distance between them in H. We compute the u-u' shortest path in H, and denote it by P. Let the set Y contain the endpoints of P, together with the following vertices of P: for each $1 \le i \le L_r$,

(i) among all vertices v of P with $\mathrm{dist}_P(v,u) \le e^{i\varepsilon_r}$, vertex that maximizes its distance to u;
(ii) among all vertices v of P with $\mathrm{dist}_P(v,u) \ge e^{i\varepsilon_r}$, vertex that minimizes its distance to u;
(iii) among all vertices v of P with $\mathrm{dist}_P(v,u') \le e^{i\varepsilon_r}$, vertex that maximizes its distance to u';
(iv) among all vertices v of P with $\mathrm{dist}_P(v,u') \ge e^{i\varepsilon_r}$, vertex that minimizes its distance to u'.

In other words, if we think of path P as a line, and then mark, for each $1 \le j \le L_r$, the point on the line that is at distance $e^{i\varepsilon_r}$ from u, and the point on the line that is at distance $e^{i\varepsilon_r}$ from u', then set Y contains, for all marked points, the vertices of P that are closest to it from both sides. By definition, $|Y| \le 4L_r$.

We apply the procedure Split to the one-hole instance (H,U), the path set that contains a single path P and vertex set Y defined above. Let (H_1,U_1) and (H_2,U_2) be the one-hole instances we get. We then return the collection $\{(H_1,U_1),(H_2,U_2)\}$.

Analysis of the small spread case. We now show that the output of the algorithm in this case satisfies the properties required in Lemma 3.2. From the definition of procedure Split, every terminal in U continues to be a terminal in at least one instance in $\{(H_1,U_1),(H_2,U_2)\}$. Moreover, since the pair u,u' of terminals is $(3/4)$-balanced, and $|Y| \le 4L_r = r/25\log^2 r$, we get that $|U_1| \le (3/4)r + r/25\log^2 r \le (9/10)r$, and similarly $|U_2| \le (9/10)r$. Also, $|U_1|+|U_2| \le |U| + O(|Y|) \le r\cdot(1+O(L_r/r)) = r\cdot(1+O(\frac{1}{\log^2 r}))$.

We now construct an algorithm Combine that satisfies the required properties. Let (H_1',U_1) be an ε-emulator for (H_1,U_1) and let (H_2',U_2) be an ε-emulator for (H_2,U_2). The algorithm Combine simply applies the procedure Glue to instances $(H_1',U_1),(H_2',U_2)$. Let (H',U') be the one-hole instance that it outputs. It is easy to verify that $U'=U$. The algorithm Combine then returns the instance (H',U). It remains to show that the algorithm Combine satisfies the required properties. Note that instance pair $(H_1,U_1),(H_2,U_2)$ is also a valid input for procedure Glue. Let $(\hat{H},\hat{U})$ be the one-hole instance that it outputs. It is easy to verify that $\hat{U}=U$. We use the following claim.

Claim 4.4. *Instance $(\hat{H},U)$ is a $(3\varepsilon_r)$-emulator for instance (H,U).*

We provide the proof of Claim 4.4 right after we complete the analysis for the small spread case. From Claim 4.3, (H',U) is an ε-emulator for $(\hat{H},U)$. From Claim 4.4, instance $(\hat{H},U)$ is a $(3\varepsilon_r)$-emulator for instance (H,U). Altogether, (H',U) is an $(\varepsilon+3\varepsilon_r) = (\varepsilon + O(\frac{\log^4 r}{r^{0.1}}))$-emulator for (H,U).

Proof (of Claim 4.4): We will show that, for each pair u_1,u_2 of terminals in U, $\mathrm{dist}_H(u_1,u_2) \le \mathrm{dist}_{\hat{H}}(u_1,u_2) \le e^{3\varepsilon_r}\cdot \mathrm{dist}_H(u_1,u_2)$.

From the procedure Split, H_1 is the subgraph of H whose image lies in the region surrounded by the image of P and the segment of outer-boundary of H from u clockwise to u' (including the boundary), and H_2 is the subgraph of H whose image lies in the region surrounded by the image of P and the segment of outer-boundary of H from u anti-clockwise to u' (including the boundary), and path P is entirely contained in both H_1 and H_2. We denote by $\hat{H}_1$ the copy of H_1 in graph $\hat{H}$, and we define graph $\hat{H}_2$ similarly, so $V(\hat{H}_1)\cap V(\hat{H}_2) = Y$. We denote by P^1,P^2 the copies of path P in graph $\hat{H}_1,\hat{H}_2$, respectively. See Figure 3 for an illustration.

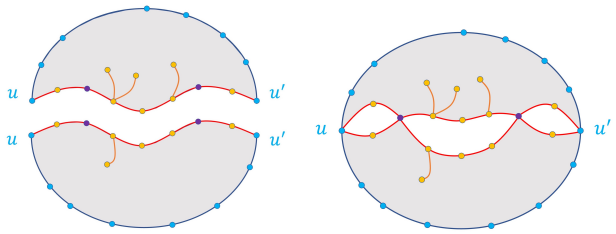


Figure 3: Illustration of graphs $\hat{H}$, H_1, and H_2. *Left:* Graphs H_1 (top) graph H_2 (bottom) viewed as individual graphs. *Right:* Subgraph $\hat{H}$ obtained by gluing graphs H_1 and H_2. Vertices in $Y\setminus\{u,u'\}$ are shown in purple.

We first show that for each pair u_1,u_2 of terminals in U one has $\mathrm{dist}_H(u_1,u_2) \le \mathrm{dist}_{\hat{H}}(u_1,u_2)$. Consider a pair $u_1,u_2 \in U$. Assume first that u_1,u_2 both belong to H_1 (the case where u_1,u_2 both belong to H_2 is symmetric). Clearly, in graph $\hat{H}$, there is a u_1-u_2 shortest path Q that lies entirely in $\hat{H}_1$. From the construction of $\hat{H}$, the same path belongs to graph H_1, and therefore $\mathrm{dist}_H(u_1,u_2) \le \mathrm{dist}_{\hat{H}}(u_1,u_2)$. Assume now that $u_1 \in V(H_1)\setminus\{u,u'\}$ and $u_2 \in V(H_2)\setminus\{u,u'\}$ (the case where $u_2 \in V(H_1)\setminus\{u,u'\}$ and $u_1 \in V(H_2)\setminus\{u,u'\}$ is symmetric). It is easy to see that, in graph $\hat{H}$, there exists a u_1-u_2 shortest path that is the sequential concatenation of

(i) a path Q_1 in $\hat{H}_1$ connecting u_1 to some vertex $x_1 \in V(P^1)$, that does not contain vertices of $V(P^1)$ as inner vertices;
(ii) a subpath R^1 of P^1 connecting x_1 to a vertex $y \in Y$;
(iii) a subpath R^2 of P^2 connecting y to a vertex x_2; and
(iv) a path Q_2 in $\hat{H}_2$ connecting x_2 to u_2 that does not contain vertices of $V(P^2)$ as inner vertices.

Consider the path in H formed by the sequential concatenation of (i) the copy of Q_1 in H_1; (ii) the subpath R of P connecting the copy of x_1 in P to the copy of x_2 in P; and (iii) the copy of Q_2 in H_2. Clearly, this path connects u_1 to u_2 in P. Moreover, since the weight of R is at most the total weight of paths R^1 and R^2, this path in H has weight at most the weight of the u_1-u_2 shortest path in $\hat{H}$. Therefore, $\mathrm{dist}_H(u_1,u_2) \le \mathrm{dist}_{\hat{H}}(u_1,u_2)$.

From now on we focus on showing that, for each pair u_1,u_2 of terminals in U, $\mathrm{dist}_{\hat{H}}(u_1,u_2) \le e^{3\varepsilon_r}\cdot \mathrm{dist}_H(u_1,u_2)$. Assume first that u_1,u_2 both belong to H_1 (the case where u_1,u_2 both belong to H_2 is symmetric). Similar to the previous discussion, the u_1-u_2 shortest path in H is entirely contained in H_1, and so $\mathrm{dist}_{\hat{H}}(u_1,u_2) = \mathrm{dist}_H(u_1,u_2)$. Assume now that $u_1 \in V(H_1)\setminus\{u,u'\}$ and $u_2 \in V(H_2)\setminus\{u,u'\}$ (the case where $u_1 \in V(H_2)\setminus\{u,u'\}$ and $u_2 \in V(H_1)\setminus\{u,u'\}$) is symmetric. Let Q be the u_1-u_2 shortest path in

H. The intersection between path Q and path P is a subpath of P. Let x_1, x_2 be the endpoints of this subpath, so vertices u_1, x_1, x_2, u_2 appear on path Q in this order. Denote by Q_1 the subpath of Q between u_1 and x_1, denote by Q_2 the subpath of Q between u_2 and x_2, and denote by Q' the subpath of Q between x_1 and x_2. We consider the following possibilities, depending on the relative locations of points x_1, x_2.

Possibility 1. There is a vertex in Y between x_1 and x_2. Let y be a vertex of Y between vertices x_1 and x_2. Consider the path $\hat{Q}$ of $\hat{H}$ formed by the sequential concatenation of (i) the copy of Q_1 in $\hat{H}_1$ connecting u_1 to the copy of x_1; (ii) the subpath R^1 of P^1 connecting the copy of x_1 to y; (iii) the subpath R^2 of P^2 connecting y to the copy of x_2; and (iv) the copy of Q_2 in $\tilde{H}_2$ connecting the copy of x_2 to u_2. Since vertex y lies between x_1 and x_2 on path P, from the construction of $\hat{H}$, the path $\hat{Q}$ in $\hat{H}$ constructed above has weight at most the weight of Q in H. Therefore, $\mathrm{dist}_{\hat{H}}(u_1, u_2) \leq \mathrm{dist}_H(u_1, u_2)$.

Possibility 2. There is no vertex of Y between x_1 and x_2. Assume without loss of generality that $|V(H_1) \cap U| \geq |U|/2$, and assume without loss of generality that x_1 is closer to u than to u' in P. We use the following observation.

Observation 4.5. $\mathrm{dist}_H(x_1, u_1) \geq \mathrm{dist}_H(x_1, u)$.

Proof: Assume not, then

$$\begin{aligned}\mathrm{dist}_H(u_1, u) &\leq \mathrm{dist}_H(x_1, u_1) + \mathrm{dist}_H(x_1, u)\\ &< 2 \cdot \mathrm{dist}_H(x_1, u) \leq \mathrm{dist}_H(u, u'), \text{ and}\\ \mathrm{dist}_H(u_1, u') &\leq \mathrm{dist}_H(x_1, u_1) + \mathrm{dist}_H(x_1, u')\\ &< \mathrm{dist}_H(x_1, u) + \mathrm{dist}_H(x_1, u') \leq \mathrm{dist}_H(u, u').\end{aligned}$$

So both $\mathrm{dist}_H(u_1, u)$ and $\mathrm{dist}_H(u_1, u')$ is less than $\mathrm{dist}_H(u, u')$. However, because $|U|/2 \leq |V(H_1) \cap U| \leq (3/4) \cdot |U|$, at least one of the pairs (u_1, u), (u_1, u') is $(3/4)$-balanced, a contradiction to the fact that u, u' is the closest $(3/4)$-balanced terminal pair in H. □

Think of path P as a line connecting u to u'. We now mark, for each $1 \leq j \leq L_r$, the point on the line that is at distance $e^{i\varepsilon_r}$ from u, and the point on the line that is at distance $e^{i\varepsilon_r}$ from u'; these marked points are called *landmarks*. Observe that there is no landmark between vertices x_1 and x_2: If there is landmark between vertices x_1 and x_2, since set Y contains, for all landmark, the vertices of P that are closest to it from both sides, either x_1 or x_2 or some other vertices of P that lie between x_1 and x_2 will be added to vertex set Y, a contradiction. Let x be the landmark closest to x_1 that lies between u and x_1, and assume $\mathrm{dist}_P(x, u) = e^{i\varepsilon_r}$. Let y be the vertex of Y closest to the landmark x that lies between x and x_1. From the construction of portals, $e^{i\varepsilon_r} \leq \mathrm{dist}_P(y, u) < \mathrm{dist}_P(x_1, u)$, $\mathrm{dist}_P(x_2, u) < e^{(i+1)\varepsilon_r}$. Therefore, $\mathrm{dist}_P(x_1, y), \mathrm{dist}_P(x_2, y) \leq (e^{\varepsilon_r} - 1) \cdot e^{i\varepsilon_r}$. Consider now the u_1-u_2 path in $\hat{H}$ formed by concatenation of (i) the copy of Q_1 in $\hat{H}_1$ connecting u_1 to the copy x_1^1 of x_1; (ii) the subpath of P^1 connecting x_1^1 to y; (iii) the subpath of P^2 connecting y to the copy x_2^2 of x_2; and (iv) the copy of Q_2 in $\hat{H}_2$ connecting x_2^2 to u_2. Calculation shows the total weight of this path is at most $e^{3\varepsilon_r} \cdot \mathrm{dist}_H(u_1, u_2)$. Therefore, $\mathrm{dist}_{\hat{H}}(u_1, u_2) \leq e^{3\varepsilon_r} \cdot \mathrm{dist}_H(u_1, u_2)$. This completes the proof of Claim 4.4. □

4.3 Large Spread Case

In this case we assume $\Phi > 2^{r^{0.9}\log^2 r}$. Without loss of generality $\min_{u,u' \in U} \mathrm{dist}_G(u, u') = 1$, so $\max_{u,u' \in U} \mathrm{dist}_G(u, u') = \Phi$. In the algorithm for this case, we use the following parameters:

$$\mu := r^2, \quad L := \lceil \log_\mu \Phi \rceil, \quad \varepsilon_r := \frac{\log^4 r}{r^{0.1}}, \quad \varepsilon'_r := \frac{1}{r^{0.7}}.$$

We first compute a hierarchical partitioning $(\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_L)$ of terminals in U in a bottom-up fashion as follows. We proceed in L iterations. In the ith iteration, we compute a collection $\mathcal{S}_i$ of subsets of U that partition U.

We start by letting collection $\mathcal{S}_0$ contain, for each terminal $u \in U$, a singleton set $\{u\}$. That is, $\mathcal{S}_0 = \{\{u\} \mid u \in U\}$. Assume we have already computed the collection $\mathcal{S}_{i-1}$ of subsets, we now describe the computation of collection $\mathcal{S}_i$, as follows. First, let graph W_{i-1} be obtained from H by contracting each subset $S \in \mathcal{S}_{i-1}$ into a single supernode, that we denote by v_S, and we define $V_{i-1} = \{v_S \mid S \in \mathcal{S}_{i-1}\}$. Then we construct another auxiliary graph R_{i-1} as follows. Its vertex set is V_{i-1}, and it contains an edge connecting v_S to $v_{S'}$ if and only if $\mathrm{dist}_{W_{i-1}}(v_S, v_{S'}) \leq \mu^i$, or equivalently $\mathrm{dist}_H(S, S') \leq \mu^i$. Finally, we define $\mathcal{S}_i$ to be the collection that contains, for each connected component C of graph R_{i-1}, the set $\bigcup_{v_S \in V(C)} S$. It is easy to verify that the sets in $\mathcal{S}_i$ partition U. This completes the description of the hierarchical partitioning $(\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_L)$. Clearly, collection $\mathcal{S}_L$ contains a single set U. We denote $\mathcal{S} = \bigcup_{0 \leq i \leq L} \mathcal{S}_i$. So collection $\mathcal{S}$ is a laminar family. That is, for every pair $S, S' \in \mathcal{S}$, either $S \cap S' = \emptyset$, or $S \subseteq S'$, or $S' \subseteq S$.

We use the following simple observation.

Observation 4.6. *For each set S in $\mathcal{S}_i$, $\mathrm{diam}_G(S) \leq 2r \cdot \mu^i$.*

It would be convenient for us to associate a partitioning tree τ with the hierarchical partitioning $(\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_L)$ we have computed, in a natural way as follows. Its vertex set $V(\tau)$ is $V(\tau) = V_0 \cup \ldots \cup V_L$ (recall that for each i, $V_i = \{v_S \mid S \in \mathcal{S}_i\}$, that is, V_i contains, for each set $S \in \mathcal{S}_i$, the supernode v_S representing S). We call nodes in V_i *level-i nodes* of tree τ, and we call sets in $\mathcal{S}_i$ *level-i sets*. Since $\mathcal{S}_L = \{U\}$, there is only one level-L node in τ, that we view as the root of τ. The edge set $E(\tau)$ of τ contains, for each pair $S, \hat{S}$ of sets such that $S \in \mathcal{S}_i, \hat{S} \in \mathcal{S}_{i+1}$ for some i and $S \subseteq \hat{S}$, an edge connecting v_S to $v_{\hat{S}}$, so v_S is a child node of $v_{\hat{S}}$, and in this case we also say that S is a *child set* of $\hat{S}$ and $\hat{S}$ is a *parent set* of S. It is easy to verify from the construction that τ is indeed a tree.

Observation 4.7. *Let S, S' be distinct sets in $\mathcal{S}$ with $S \cap S' = \emptyset$. Let u_1, u_2 be any pair of vertices in S, and let u'_1, u'_2 be any pair of vertices in S'. Then the pairs (u_1, u_2) and (u'_1, u'_2) are non-crossing.*

Proof: Assume for contradiction that the pairs (u_1, u_2) and (u'_1, u'_2) are crossing. Assume that S is a level-i set and S' is a level-i' set, and assume without loss of generality that $i \geq i'$.

We first find another two pairs $(u_3, u_4), (u'_3, u'_4)$ of terminals in S such that $\mathrm{dist}_G(u_3, u_4) \leq \mu^i$, $\mathrm{dist}_G(u'_3, u'_4) \leq \mu^{i'}$ and the pairs (u_3, u_4) and (u'_3, u'_4) are crossing. We start by finding the pair (u_3, u_4). In fact, if we denote by γ_1 the boundary segment clockwise from u'_1 to u'_2 around the outerface of H, and denote by γ_2 the boundary segment clockwise from u'_2 to u'_1 around the outerface of H, then since we have assumed that (u_1, u_2) and (u'_1, u'_2) are

crossing, one of u_1, u_2 lies on γ_1 and the other lies on γ_2. Assume without loss of generality that u_1 lies on γ_1 and u_2 lies on γ_2.

From the construction of graphs $R_1, \ldots, R_{i-1}$ and collections $\mathcal{S}_1, \ldots, \mathcal{S}_i$. It is easy to observe that, for every pair u, u' of terminals that belong to the same level-i set, there exists a sequence $u^1, \ldots, u^t$ of terminals in U that all belong to the same level-i set as u and u', such that, if we denote $u = u^0$ and $u' = u^{t+1}$, then for each $0 \le j \le t$, $\mathrm{dist}_G(u^j, u^{j+1}) \le \mu^i$; and for every pair u, u' of terminals do not belong to the same level-i set, $\mathrm{dist}_G(u, u') > \mu^i$.

Consider now the pair u_1, u_2 of terminals. Note that they belong to the same level-i set. From the above discussion, there exists a sequence of terminals in S starting with u_1 and ending with u_2, such that the distance between every pair of consecutive terminals in the sequence is less than μ^i. Since u_1 lies on γ_1 and u_2 lies on γ_2, there must exist a pair (u_3, u_4) of terminals appearing consecutively in the sequence, such that u_3 lies on γ_1 and u_4 lies on γ_2, so pairs (u_3, u_4) and (u_1', u_2') are crossing and $\mathrm{dist}_G(u_3, u_4) \le \mu^i$.

We can then use similar arguments to find another pair (u_3', u_4') crossing the pair (u_3, u_4) and $\mathrm{dist}_G(u_3', u_4') \le \mu^{i'}$. Note that, since $u_3, u_4 \in S$ and $u_3', u_4' \notin S$, $\mathrm{dist}_G(u_3, u_3') > \mu^i$ and $\mathrm{dist}_G(u_4, u_4') > \mu^i$. Altogether, we get that

$$\begin{aligned}\mathrm{dist}_G(u_3', u_4') + \mathrm{dist}_G(u_3, u_4) &\le \mu^i + \mu^{i'} \le \mu^i + \mu^i \\ &< \mathrm{dist}_G(u_3, u_3') + \mathrm{dist}_G(u_4, u_4'),\end{aligned}$$

a contradiction to the Monge property on the crossing pairs (u_3, u_4) and (u_3', u_4'). □

Expanding sets. The central notion in the algorithm for the large spread case is the notion of *expanding sets*. Recall that $\varepsilon_r' = \frac{1}{r^{0.7}}$. We say that a set $S \in \mathcal{S}$ is *expanding*, if and only if $|\hat{S}| \ge e^{\varepsilon_r'} \cdot |S|$, where $\hat{S}$ is the parent set of S (or equivalently $v_{\hat{S}}$ is the parent node of v_S in τ), otherwise we say it is *non-expanding*. We now distinguish between two cases, depending on whether or not $\mathcal{S}$ contains a non-expanding set with moderate size.

4.3.1 Balanced Case: There are non-expanding S such that $r/5 \le |S| \le 4r/5$. We let $\hat{S}$ be the parent set of S. We denote $S^* = \hat{S} \setminus S$, and $S' = U \setminus \hat{S}$, so sets S^*, S, S' partition set U. Moreover, we have $r/6 \le |S|, |S'| \le 5r/6$ and $|S^*| \le (e^{\varepsilon_r'} - 1)r$.

We will employ the procedure Split in order to decompose the instance (H, U) into smaller instances, for which we need to compute a non-crossing path set and a set of vertices in the path set, as the input to the procedure, as follows.

We say that an ordered pair (u, u') of terminals in S is a *border pair* if walking on the outer-boundary of H from u clockwise to u' contains no other vertices of S but at least one vertex of $S^* \cup S'$. We compute the set $\mathcal{M}$ of all border pairs in S, and then apply the algorithm from Lemma 2.2 to graph H and the set $\mathcal{M}$ of pairs, and obtain a set $\mathcal{P}$ of shortest paths connecting pairs in $\mathcal{M}$. We call $\mathcal{P}$ the *border path set* of S. It is easy to verify that set $\mathcal{M}$ is non-crossing, and so path set $\mathcal{P}$ is also non-crossing.

Consider now a border pair (u, u') of terminals and let $P_{(u,u')}$ be the u-u' shortest path that we have computed. We apply the algorithm from Lemma 2.4 to graph H, path $P_{(u,u')}$ and each vertex $u^* \in S^*$ that lies on the outer-boundary of H from u clockwise to u' with parameter ε_r, and compute an ε_r-cover of u^* on $P_{(u,u')}$. We then let $Y_{(u,u')}$ be the union of all such ε_r-covers and the endpoints of $P_{(u,u')}$, so $Y_{(u,u')}$ is a vertex set of $P_{(u,u')}$. Denote $Y = \bigcup_{(u,u')} Y_{u,u'}$, so Y is a vertex set of $V(\mathcal{P})$ that contains all endpoints of paths in $\mathcal{P}$. Moreover, from Lemma 2.4,

$$|Y \setminus U| \le O\left(\frac{|S^*|}{\varepsilon_r}\right) \le O\left(\frac{(e^{\varepsilon_r'} - 1) \cdot r}{\varepsilon_r}\right) = O\left(\frac{r^{0.4}}{\log^4 r}\right).$$

We then apply the procedure Split to the one-hole instance (H, U), the non-crossing path set $\mathcal{P}$, and the vertex set Y. We return the collection $\mathcal{H}$ of one-hole instances output by the procedure Split as the output of our algorithm in this case.

Analysis of Balanced Case. We now show that the output collection of one-hole instances of the above algorithm satisfies the properties required in Lemma 3.2.

First, from the construction of the border path set $\mathcal{P}$, the instances in $\mathcal{H}$ can be partitioned into two subsets: $\mathcal{H}_1$ contains all instances that corresponds to a region in H surrounded by a segment of outer-boundary of H and the image of some path $P \in \mathcal{P}$; and set $\mathcal{H}_2$ contains all other instances. Recall that $r/6 \le |S|, |S'| \le 5r/6$ and $|Y| \le O(r^{0.4}/\log^4 r)$. On the one hand, each instance in $\mathcal{H}_1$ contains at most two terminals in S, and so it contains at most $r - |S| + 2 + |Y| \le (9/10)r$ terminals. On the other hand, each instance in $\mathcal{H}_2$ does not contain terminals in S', and so it contains at most $r - |S'| + |Y| \le (9/10)r$ terminals.

Second, note that $|Y| \le O(\frac{r^{0.4}}{\log^4 r})$ and $\lambda = c^* \log^2 k/\varepsilon^{20}$, then from Claim 4.2 (by setting $\mu = \lambda$), we get that $\sum_{(H_i,U_i)\in\mathcal{H}} |U_i| \le O(r \log^2 r)$ and $\sum_{(H_i,U_i)\in\mathcal{H}:|U_i|>\lambda} |U_i| \le |U| \cdot \left(1 + O(\frac{1}{\log^2 r})\right)$.

We now construct an algorithm Combine that satisfies the required properties. Recall that we are given, for each $(H_i, U_i) \in \mathcal{H}$, an ε-emulator (H_i', U_i) for instance (H_i, U_i). The algorithm Combine simply applies the procedure Glue to $(H_1', U_1), \ldots, (H_s', U_s)$ and let (H', U) be the output one-hole instance that it outputs. The algorithm Combine simply returns instance (H', U). It remains to show that the algorithm Combine satisfies the required properties. Note that one-hole instances $(H_1, U_1), \ldots, (H_s, U_s)$ is also a valid input for procedure Glue. Let $(\hat{H}, \hat{U})$ be the one-hole instance that the procedure Glue outputs when it is applied to instances $(H_1, U_1), \ldots, (H_s, U_s)$. It is easy to verify that $\hat{U} = U$. We use the following claim whose proof is in the full version.

Claim 4.8. *Instance $(\hat{H}, U)$ is an $O(\varepsilon_r)$-emulator for (H, U).*

Now we complete the proof of Lemma 3.2 for the Balanced Case using Claim 4.8. In fact, since for each $1 \le i \le t$, (H_i', U_i) is an ε-emulator for (H_i, U_i), from Claim 4.3, (H', U) is an ε-emulator for $(\hat{H}, U)$. Then from Claim 4.4 and Claim 4.8, we get that (H', U) is an $(\varepsilon + O(\varepsilon_r)) = (\varepsilon + O(\frac{\log^4 r}{r^{0.1}}))$-emulator for (H, U). Moreover, from the algorithm Glue, the number of vertices in graph H' that does not belong to graphs $H_1', \ldots, H_s'$ are the branch vertices in graph H (vertices that appear on the boundary of at least three regions when applying the procedure Split to it), and from Claim 4.1 the number of such vertices is bounded by $O(r \log^2 r)$. Therefore, $|V(H')| \le \left(\sum_{1\le i\le s} |V(H_i')|\right) + O(\log^2 r \cdot |U|)$.

4.3.2 Unbalanced Case: Every S is either expanding, or $|S| < r/5$, or $|S| > 4r/5$.

Step 1: Reducing the spread from above. We say that a set $S \in \mathcal{S}$ is *heavy* if and only if $|S| > 4r/5$, and in this case we also say that

the node v_S is heavy. Clearly, every level of τ contains at most one heavy node, and the set of heavy nodes induce a path in τ which ends at the root node of τ. Let $\hat{S}$ be the non-expanding heavy set that lies on the lowest level. We denote by $\tilde{L}$ the level that $\hat{S}$ lies in and let $\check{S}$ be its parent set. Define $\hat{S}^* = \check{S} \setminus \hat{S}$ and $\hat{S}' = U \setminus \check{S}$. So sets $\hat{S}^*, \hat{S}, \hat{S}'$ partition set U, and $|\hat{S}^*| \leq (e^{\varepsilon'_r} - 1)r$. We perform the same operations as in the Balanced Case (Section 4.3.1) to graph H with respect to the partition $(\hat{S}, \hat{S}^*, \hat{S}')$. Let $\hat{\mathcal{H}}$ be the collection of instances we obtain. From similar analysis as in Section 4.3.1, we get that $\sum_{(H_i,U_i)\in\hat{\mathcal{H}}} |U_i| \leq O(r \log^2 r)$, and $\sum_{(H_i,U_i)\in\hat{\mathcal{H}}:|U_i|>\lambda} |U_i| \leq r \cdot \left(1 + O(\frac{1}{\log^2 r})\right)$. If additionally we have, for each $(H_i, U_i) \in \hat{\mathcal{H}}$, $|U_i| \leq 0.9r$, then we simply return the collection $\hat{\mathcal{H}}$ as the output. Assume now that there exists some instance $(H_{i^*}, U_{i^*}) \in \hat{\mathcal{H}}$ with $|U_{i^*}| > 0.9r$. Since $\sum_{(H_i,U_i)\in\hat{\mathcal{H}}:|U_i|>\lambda} |U_i| \leq r \cdot \left(1 + O(\frac{1}{\log^2 r})\right)$, we may have only one such instance. It is easy to see from the algorithm Split that no terminal of U_{i^*} is a cut vertex in graph H_{i^*}. Note that it is now enough to prove Lemma 3.2 for the instance (H_{i^*}, U_{i^*}), which we do in the second step. Indeed, if Lemma 3.2 holds for instance (H_{i^*}, U_{i^*}), then we simply apply the algorithm from Lemma 3.2 to instance (H_{i^*}, U_{i^*}) and obtain a collection $\mathcal{H}^*$ instances. We simply return the collection $\tilde{\mathcal{H}} = (\hat{\mathcal{H}} \setminus \{(H_{i^*}, U_{i^*})\}) \cup \mathcal{H}^*$. It is easy to verify that the output collection $\tilde{\mathcal{H}}$ satisfies all conditions in Lemma 3.2 for the original input instance (H, U).

Step 2: Reducing the spread from below. The goal of this final step is to further modify and decompose the instance (H_{i^*}, U_{i^*}) into instances with bounded spread, and eventually apply the algorithm from Case 1 to them. Consider the instance (H_{i^*}, U_{i^*}). From the algorithm Split, the instance (H_{i^*}, U_{i^*}) corresponds to a region of graph H, that is surrounded by shortest paths connecting terminals in U. Therefore, for every pair v, v' of vertices in H_{i^*} (that are also vertices in H), $\mathrm{dist}_H(v, v') = \mathrm{dist}_{H_{i^*}}(v, v')$. Note that set U_{i^*} can be partitioned into two subsets: set $\tilde{S}$ contains all terminals in $\hat{S}$ that lies in U_{i^*}, and set Y_{i^*} contains all new terminals (which are vertices in ε_r-covers of vertices of $\hat{S}^*$ on paths of $\mathcal{P}$) added in Step 1 that lie on the boundary of graph H_{i^*}. Note that the distances between a pair of terminals in Y_{i^*} and the distances between a terminal in Y_{i^*} and a terminal in $\tilde{S}$ could be very small (even much smaller than $\min_{u,u'} \mathrm{dist}_H(u, u')$) at the moment, which makes it hard to bound the spread from above. Therefore, we start by modifying the instance (H_{i^*}, U_{i^*}) as follows.

We let graph $\tilde{H}$ be obtained from H_{i^*} by adding, for each terminal $u \in Y_{i^*}$, a new vertex $\tilde{u}$ and an edge $(\tilde{u}, u)$ with weight $\mu^{\tilde{L}-1}$. We then define $\tilde{U} = \tilde{S} \cup \{\tilde{u} \mid u \in Y_{i^*}\}$. This completes the construction of the new instance $(\tilde{H}, \tilde{U})$. We call this operation *terminal pulling*. It is easy to verify that $(\tilde{H}, \tilde{U})$ is a one-hole instance, and moreover, for each new terminal in $\tilde{U} \setminus \tilde{S}$, the distance in $\tilde{H}$ from it to any other terminal in $\tilde{U}$ is at least $\mu^{\tilde{L}-1}$. We denote $\tilde{Y} = \{\tilde{u} \mid u \in Y_{i^*}\}$, so $\tilde{U} = \tilde{S} \cup \tilde{Y}$. We will show later in the analysis that it is now sufficient to prove Lemma 3.2 for the instance $(\tilde{H}, \tilde{U})$.

We now construct the hierarchical clustering $\tilde{\mathcal{S}}$ for instance $(\tilde{H}, \tilde{U})$, in the same way as the hierarchical clustering $\mathcal{S}$ for instance (H, U), described at the beginning of the large spread case. Let $\tilde{\tau}$ be the partitioning tree associated with $\tilde{\mathcal{S}}$. Recall that for every pair of vertices in H_{i^*}, the distance between them in H_{i^*} is identical to the distance between them in H. From the construction of instance $(\tilde{H}, \tilde{U})$, it is easy to verify that both $\tilde{\mathcal{S}}$ and $\tilde{\tau}$ has depth $\tilde{L}$, and in levels $\tilde{L} - 1, \ldots, 1$, new terminals in $\tilde{U} \setminus \tilde{S}$ only form singleton sets as each of them is at distance at least $\mu^{\tilde{L}-1}$ from any other terminal in $\tilde{U}$. Therefore, every non-singleton set in $\tilde{\mathcal{S}}$ is also a set in $\mathcal{S}$.

We say that a set is *good* if

(i) $|S| > 1$;
(ii) S lies on level at most $\tilde{L} - 2\log r/\varepsilon'_r$;
(iii) S is non-expanding; and
(iv) for any other set $S' \in \tilde{\mathcal{S}}$ that lies on level at most $\tilde{L} - 2\log r/\varepsilon'_r$ and $S \subseteq S'$, S' is expanding.

We denote by $\tilde{\mathcal{S}}_g$ the collection of all good sets in $\tilde{\mathcal{S}}$. We prove in the following observation that all good sets in $\tilde{\mathcal{S}}_g$ lie on level at least $\tilde{L} - O(\log r/\varepsilon'_r)$. From property (ii), and our assumption for Case 2 that any set $S \in \mathcal{S}$ with $r/5 \leq |S| \leq 4r/5$ is expanding, it is easy to see that all good sets S have size at most $r/5$ (we have used the property that every non-singleton set in $\tilde{\mathcal{S}}$ is also a set in $\mathcal{S}$).

Observation 4.9. *All good sets in $\tilde{\mathcal{S}}_g$ lie on level at least $\tilde{L} - 10\log r/\varepsilon'_r$. Every terminal either forms a singleton set on level at least $\tilde{L} - 10\log r/\varepsilon'_r$, or belongs to some good set in $\tilde{\mathcal{S}}_g$.*

Now for each good set S, we compute its border path set $\tilde{\mathcal{P}}_S$ in instance $(\tilde{H}, \tilde{U})$ in the same way as in the Balanced Case (Section 4.3.1). Now define $\tilde{\mathcal{P}} = \bigcup_{S\in\tilde{\mathcal{S}}_g} \tilde{\mathcal{P}}_S$. We show in the next observation that the collection $\mathcal{P}$ of paths is non-crossing.

Observation 4.10. *The collection $\tilde{\mathcal{P}}$ of paths is non-crossing.*

Proof: Assume for contradiction that the collection $\tilde{\mathcal{P}}$ of paths is not non-crossing. Then there exist two distinct sets $S, S' \in \tilde{\mathcal{S}}_g$, a border path P connecting terminals u_1, u_2 in S and a border path P' of S' connecting terminals u'_1, u'_2 in S', such that the pairs $(u_1, u_2), (u'_1, u'_2)$ are crossing. However, from the definition of good sets, $S \cap S' = \emptyset$. Therefore, from Observation 4.7, pairs $(u_1, u_2), (u'_1, u'_2)$ are non-crossing, a contradiction. □

Consider now a good set $S \in \tilde{\mathcal{S}}_g$. We define $S^* = \check{S} \setminus S$, where $\check{S}$ is the parent set of S in $\tilde{\mathcal{S}}_g$. Recall that a pair (u, u') of terminals in S is a border pair, if and only if the outer-boundary of $\tilde{H}$ connecting u to u' contains no other vertices of S but at least one vertex that does not lie in S. Now for each border pair (u, u') of terminals in S, let $P_{(u,u')}$ be the u-u' shortest path in $\tilde{\mathcal{P}}_S$ that we have computed. We apply the algorithm from Lemma 2.4 to each vertex $u^* \in S^*$ that lies on the outer-boundary from u clockwise to u' with parameter ε_r, and compute an ε_r-cover of u^* on $P_{(u,u')}$. We then let $Y^S_{(u,u')}$ be the union of all such ε_r-covers and the endpoints of $P_{(u,u')}$. We then let set Y^S be the union of the sets $Y^S_{(u,u')}$ for all border pairs (u, u'). Finally, we set $Y = \bigcup_{S\in\tilde{\mathcal{S}}_g} Y^S$, so Y is a vertex set of $V(\tilde{\mathcal{P}})$ that contains all endpoints of paths in $\tilde{\mathcal{P}}$. Moreover, from Lemma 2.4,

$$|Y| \leq O\left(\sum_{S\in\tilde{\mathcal{S}}_g} \frac{|S^*|}{\varepsilon_r}\right) \leq O\left(\frac{(e^{\varepsilon'_r} - 1) \cdot \sum_{S\in\tilde{\mathcal{S}}_g} |S|}{\varepsilon_r}\right)$$
$$\leq O\left(\frac{(e^{\varepsilon'_r} - 1) \cdot r}{\varepsilon_r}\right) = O\left(\frac{(1/r^{0.7}) \cdot r}{\log^4 r/r^{0.1}}\right) = O\left(\frac{r^{0.4}}{\log^4 r}\right).$$

We now apply the algorithm Split to instance $(\tilde{H},\tilde{U})$, the path set $\tilde{\mathcal{P}}$ and the vertex set Y. Let $\tilde{\mathcal{H}}$ be the collection of one-hole instances we get. If all instances $(\hat{H},\hat{U})$ in $\tilde{\mathcal{H}}$ satisfy that $|\hat{U}|\leq 0.9r$, then we terminate the algorithm and return $\tilde{\mathcal{H}}$. Assume that there is some instance $(\hat{H},\hat{U})$ in $\tilde{\mathcal{H}}$ such that $|\hat{U}|>0.9r$. From similar analysis in Step 1, there can be at most one such instance. We denote such an instance by $(\hat{H},\hat{U})$.

From the algorithm Split, $(\hat{H},\hat{U})$ corresponds to a region of $\tilde{H}$ surrounded by segments of paths in $\tilde{\mathcal{P}}$. Let $Q_1,\ldots,Q_k$ be the segments of the boundary of $\hat{H}$. If there are two segments Q_i,Q_j that are subpaths of shortest paths in the same set $\tilde{\mathcal{P}}_S$, then from the definition of border pairs and border path sets, $\hat{U}$ may not contain terminals in any good set of $\tilde{\mathcal{S}}_g$ other than S. However, since $|S|\leq r/5$ and $|Y|\leq O\big(\frac{r^{0.4}}{\log^4 r}\big)$, this contradicts to the assumption that $|\hat{U}|>0.9r$. Therefore, for each set $S\in\tilde{\mathcal{S}}_g$, the boundary of $\hat{H}$ contains at most one segment of some border path of $\tilde{\mathcal{P}}_S$.

We now modify the instance $(\hat{H},\hat{U})$ as follows. Denote $L^*=\tilde{L}-10\log r/\varepsilon_r'$. Let H^* be the graph obtained from $\hat{H}$ by applying the terminal pulling operation to every terminal in $\hat{U}\setminus\tilde{S}$ via an edge of weight μ^{L^*-1}. We then define set U^* to be the union of $(\hat{U}\cap\tilde{S})$ and the set of all new terminals created in the terminal pulling operation. We use the following observation.

Observation 4.11. $\Phi(H^*,U^*)\leq 2^{O(\log^2 r/\varepsilon_r')}$.

Proof: From Observation 4.9, every pair of terminals in U^* has distance at least μ^{L^*-1} in graph H^*. On the other hand, since $\hat{H}$ is a subgraph of $\tilde{H}$, every terminal pair in U^* has distance at most $\mu^{\tilde{L}+1}$ in H^*. Therefore, $\Phi(H^*,U^*)\leq\mu^{\tilde{L}-L^*+2}=2^{O(\log^2 r/\varepsilon_r')}$. □

Since $2^{O(\log^2 r/\varepsilon_r')}<2^{r^{0.9}\log^2 r}$ when r is larger than some large enough constant, we apply the algorithm from Case 1 to (H^*,U^*) and obtain a collection of instances $\mathcal{H}(\hat{H},\hat{U})$. The output of the algorithm is the collection $\big(\tilde{\mathcal{H}}\setminus\{(\hat{H},\hat{U})\}\big)\cup\mathcal{H}(\hat{H},\hat{U})$.

Analysis of Unbalanced Case. Recall that in this step we assume that, after Step 1, there is an instance (H_{i^*},U_{i^*}) with $|U_{i^*}|>0.9r$, and we transformed it into another instance $(\tilde{H},\tilde{U})$. We first show that it is sufficient to prove Lemma 3.2 for instance $(\tilde{H},\tilde{U})$. All other conditions can be easily verified. We now show that when applying the algorithm Glue to ε-emulators $\{(\tilde{H}',\tilde{U})\}\cup\{(H_i',U_i)\}_{i\neq i^*}$, we still obtain an $(\varepsilon+O(\frac{\log^4 r}{r^{0.1}}))$-emulator for (H,U). In fact, we only need to consider the terminal pairs u,u' with $u\in S$ and $u'\notin S$. Note that such a pair u,u' of terminals belongs to different level-$\tilde{L}$ clusters in $\mathcal{S}$. From the construction of $\tilde{\mathcal{S}}$, $\mathrm{dist}_H(u,u')\geq\mu^{\tilde{L}}$. Therefore, the transformation from instance (H_{i^*},U_{i^*}) to instance $(\tilde{H},\tilde{U})$ adds at most an additive $\mu^{\tilde{L}-1}$ to their distance, which is at most $O(\frac{1}{\mu})=O(\frac{1}{r^2})\leq O(\frac{\log^4 r}{r^{0.1}})$-fraction of their distance in graph H. Therefore, by gluing the ε-emulators $\{(\tilde{H}',\tilde{U})\}\cup\{(H_i',U_i)\}_{i\neq i^*}$, we still obtain an $(\varepsilon+O(\frac{\log^4 r}{r^{0.1}}))$-emulator for (H,U).

From now on, we focus on proving that the decomposition we computed for instance $(\tilde{H},\tilde{U})$ satisfies all properties in Lemma 3.2. Recall that we have first computed a collection $\tilde{\mathcal{S}}_g$ of good sets, computed a path set $\tilde{\mathcal{P}}$ and a subset Y of vertices in $V(\tilde{\mathcal{P}})$ based on sets in $\tilde{\mathcal{S}}_g$, and then applied the procedure Split to $((\tilde{H},\tilde{U}),\tilde{\mathcal{P}},Y)$ and obtained a collection $\tilde{\mathcal{H}}$ of one-hole instances.

Assume first that all instances $(\hat{H},\hat{U})$ in collection $\tilde{\mathcal{H}}$ satisfy that $|\hat{U}|\leq 0.9r$. Since $|Y|\leq O\big(\frac{r^{0.4}}{\log^4 r}\big)$, from Claim 4.2, we get that $\sum_{(\hat{H},\hat{U})\in\tilde{\mathcal{H}}}|\hat{U}|\leq O(r\log^2 r)$ and $\sum_{(\hat{H},\hat{U})\in\tilde{\mathcal{H}}:|\hat{U}|>\lambda}|\hat{U}|\leq r\cdot\big(1+O(\frac{1}{\log^2 r})\big)$. We now describe the algorithm Combine that, takes as input, for each instance $(\hat{H},\hat{U})\in\mathcal{H}$, an ε-emulator $(\hat{H}',\hat{U})$, computes an $\big(\varepsilon+O(\varepsilon_r)\big)=\big(\varepsilon+O(\frac{\log^4 r}{r^{0.1}})\big)$-emulator for $(\tilde{H},\tilde{U})$. We simply apply the algorithm Glue to instances $\{(\hat{H}',\hat{U})\mid(\hat{H},\hat{U})\in\tilde{\mathcal{H}}\}$ and denote the obtained instance by $(\tilde{H}',\tilde{U})$. The proof that instance $(\tilde{H}',\tilde{U})$ is indeed a $\big(\varepsilon+O(\varepsilon_r)\big)$-emulator for $(\tilde{H},\tilde{U})$ and the proof that $|V(\tilde{H}')|\leq\big(\sum_{1\leq i\leq s}|V(\hat{H}_i')|\big)+O(\log^2 r\cdot|U|)$ use identical arguments in the Balanced Case, and is omitted here.

Assume now that there exists an instance $(\hat{H},\hat{U})$ in collection $\tilde{\mathcal{H}}$ with $|\hat{U}|>0.9r$. Denote $\tilde{\mathcal{H}}'=\tilde{\mathcal{H}}\setminus\{(\hat{H},\hat{U})\}$ and denote by $\overline{\mathcal{H}}=\big(\tilde{\mathcal{H}}\setminus\{(\hat{H},\hat{U})\}\big)\cup\mathcal{H}(\hat{H},\hat{U})$ the output collection of instances. First, note that all instances $(\overline{H},\overline{U})$ in collection $\tilde{\mathcal{H}}'$ satisfy that $|\overline{U}|\leq 0.9r$. Since the remaining instances in $\overline{\mathcal{H}}$ are obtained by applying the algorithm from Case 1 to the instance (H^*,U^*), that is obtained from modifying the unique large instance in $(\hat{H},\hat{U})$. From the algorithm in Case 1, we know that each instance in the output collection contains at most $0.9r$ terminals. Second, from similar arguments, we get that $\sum_{(\overline{H},\overline{U})\in\overline{\mathcal{H}}}|\overline{U}|\leq O(r\log^2 r)$ and $\sum_{(\overline{H},\overline{U})\in\overline{\mathcal{H}}:|\overline{U}|>\lambda}|\overline{U}|\leq r\cdot\big(1+O(\frac{1}{\log^2 r})\big)$. We now describe the algorithm Combine that, takes as input, for each instance $(\overline{H},\overline{U})\in\mathcal{H}$, an ε-emulator $(\overline{H}',\overline{U})$, computes an $\big(\varepsilon+O(\varepsilon_r)\big)=\big(\varepsilon+O(\frac{\log^4 r}{r^{0.1}})\big)$-emulator for $(\tilde{H},\tilde{U})$. First, consider the instances in $\mathcal{H}(\hat{H},\hat{U})$ that are obtained from applying the algorithm in Case 1 to (H^*,U^*). We simply use the algorithm Combine described in Case 1 to compute an $\big(\varepsilon+O(\varepsilon_r)\big)$-emulator (H^{**},U^*) for instance (H^*,U^*). Finally, we apply the algorithm Glue to instances in $\{(\overline{H}',\overline{U})\mid(\overline{H},\overline{U})\in\tilde{\mathcal{H}}'\}\cup\{(H^{**},U^*)\}$ and denote the obtained instance by $(\tilde{H}',\tilde{U})$. Note that, for different sets $S,S'\in\tilde{\mathcal{S}}_g$ such that $S\cap\hat{U}\neq\emptyset,S'\cap\hat{U}\neq\emptyset$ and $S\cap S=\emptyset$, if set S lies on level i and set S' lies on level i', then $\mathrm{dist}(S,S')\geq\mu^{(\max\{i,i'\}+1)}\geq\mu^{L^*}$. Therefore, from similar arguments at the beginning of the analysis, the terminal pulling operation only incurs a multiplicative factor-$O(1/r)$ error of the distances between terminals in disjoint sets in $\tilde{\mathcal{S}}_g$.

The rest of the proof that instance $(\tilde{H}',\tilde{U})$ is indeed an $\big(\varepsilon+O(\varepsilon_r)\big)$-emulator for $(\tilde{H},\tilde{U})$ uses almost identical arguments in the Balanced Case, and is omitted here.

5 EMULATOR FOR EDGE-WEIGHTED PLANAR GRAPHS, AND APPLICATIONS

In this section we sketch a proof of Theorem 1.1, followed by a bootstrapping idea so that an ε-emulator for any $O(1)$-hole instance of size $O_\varepsilon(k\,\mathrm{polylog}\,k)$ can be computed in $O_\varepsilon(n\,\mathrm{poly}(\log^* n))$ time. Using this extremely fast emulator construction we obtain new efficient algorithms for the optimization problems on planar graphs.

Intuitively, our algorithm first computes an $O(n/k)$-division of graph G using the algorithm from Lemma 5.1, reducing it to a

collection of $O(1)$-hole instances each containing $O(1)$ many terminals in T. For each instance we further decompose it into one-hole instances. Compute an ε-emulator for each one-hole instance using Theorem 3.1, and finally glue all these ε-emulators together to get the ε-emulator for the input instance (G, T). Due to space constraint, we defer the complete proof to the full version.

Separators and recursive decomposition. For any $r \geq 0$, an *r-division* [23] of a planar graph G is a decomposition of G into edge-disjoint subgraphs of G, called the *pieces*, such that

- there are at most $O(|V(G)|/r)$ pieces,
- each piece has size at most r,
- each piece has at most $O(\sqrt{r})$ *boundary vertices*, where a *boundary vertex* is one that belongs to multiple pieces, and
- the number of holes in each piece is bounded by $O(1)$.

An r-division can be computed in linear time [36]. For our application we want the r-division to evenly distribute the terminals among the pieces as well.

Lemma 5.1. *Given a planar graph P with n vertices and t terminals, one can compute in $O(n)$-time an r-division for P such that each piece has $O(tr/n)$ terminals.*

Given the pieces each containing $O(1)$ holes, we use a split-and-glue strategy similar to the one in Section 4.1 to reduce the problem to one-hole instances, and apply Theorem 3.1 to compute ε-emulator for each one-hole instance. We summarize the construction with the following lemma.

Lemma 5.2. *For any parameter $\varepsilon \in (0, 1)$ and any h-hole instance (H, U) with $n := |H|$ and $r := |U|$, there is another h-hole instance (H', U) that is an ε-emulator for (H, U), such that $|V(H')| \leq \tilde{O}(r) \cdot f(h) \cdot \varepsilon^{-O(h)}$ for some function $f(h)$. The running time of the algorithm is $(n + r^2) \cdot (h \log n/\varepsilon)^{O(h)}$.*

Replace the input general planar graph G with an ε-emulator of size $N := O(k^2 \log^2 n/\varepsilon^2)$ using a slight modification of the construction of Cheung, Goranci, and Henzinger [13, Theorem 6.9] (by removing the processing step of reducing the graph size to $O(k^4)$). Apply r-division from Lemma 5.1 to G for $r := O(N/k) = \tilde{O}(k/\varepsilon^2)$ to even distribute the vertices, boundary vertices, holes, and terminals into the pieces. Each piece in the r-division now has $r = O(N/k)$ vertices, $\sqrt{r} = O(N^{1/2}/k^{1/2})$ boundary vertices, $O(1)$ number of holes, and $O(1)$ number of terminals. Now apply Lemma 5.2 on each piece P to obtain a planar emulator on the set of boundary vertices and terminals in P. Stitching the emulators for all the pieces together into a new graph G', which has size

$$O\left(\frac{N}{N/k} \cdot \frac{N^{1/2}}{\operatorname{poly}\varepsilon \cdot k^{1/2}}\right) = O\left(\frac{N^{1/2} \cdot k^{1/2}}{\operatorname{poly}\varepsilon}\right) = O\left(\frac{k^{3/2}}{\operatorname{poly}\varepsilon}\right).$$

Graph G' is in fact a planar emulator of G with respect to the terminals T. The construction of ε-emulator by Cheung-Goranci-Henzinger takes $O(n \log n/\varepsilon)$ time (without the preprocessing). Constructing the emulators for the pieces using Lemma 5.2 takes time $(N/r) \cdot r(\log r/\varepsilon)^{O(1)} \leq N \cdot (\log N/\varepsilon)^{O(1)}$. Therefore the running time is $n \log^{O(1)} n/\operatorname{poly}\varepsilon$ overall.

We can further bootstrap the construction to reduce the size of the emulator to $k^{1+o(1)}/\operatorname{poly}\varepsilon$, and derive Theorem 1.1. On the other hand, we can also bootstrap the construction time when the input is an $O(1)$-hole instance.

Theorem 5.3. *Given any parameter $0 < \varepsilon < 1$ and any planar graph P with n vertices and k terminals where $k \leq \sqrt{n}$ all lying on a constant number of faces of P, one can compute an ε-emulator of P with respect to the terminals of size $O(k \operatorname{polylog} k/\operatorname{poly}\varepsilon)$ in $O(n \log^{(c)} n/\operatorname{poly}\varepsilon + cn)$ time for any integer $c \geq 1$.*[5]

Proof (sketch): To get an intuition of the proof, let's look at the case when $c = 2$, and ignore the restriction on the number of terminals and the ε-factors in the running time for a moment.

- First compute r-division of P for $r = (\log\log n)^{4C}$, where C is bigger than the number of logs we need in the running time of Lemma 5.2. Replace each piece in the r-division by an ε-emulator with respect to the boundary vertices and terminals using Lemma 5.2; the total time on the emulator construction is $O_\varepsilon(n(\log\log\log n)^{O(1)})$ and the new graph P' has size $O_\varepsilon(n/(\log\log n)^{2C})$.
- Now the graph is $(\log\log n)^{2C}$-factor smaller than original, we can compute another r'-division for $r' = (\log n)^{4C}$, and replace each piece in the r'-division by an ε-emulator with respect to the boundary vertices and terminals. This way, instead of spending $O_\varepsilon(n \operatorname{poly}\log\log n)$ time if we perform r'-division directly on the original graph, now it takes
$$O_\varepsilon\left(\frac{n}{(\log\log n)^{2C}} \cdot (\log\log n)^C\right)$$
time. The new graph P'' has size $n/(\log n)^{2C}$.
- Finally, compute an ε-emulator for P'' with respect to the terminals, which takes
$$O_\varepsilon\left(\frac{n}{(\log n)^{2C}} \cdot (\log n)^C\right) = O_\varepsilon(n/(\log n)^C)$$
time. The final emulator has size $O_\varepsilon(k \operatorname{polylog} k)$.
- Overall the bottleneck is the compute the first set of emulators in the r-division, which takes $O(n \operatorname{poly}\log\log\log n)$ time. The accumulated distortion in distance is 3ε. □

By taking $c = \log^* n$, we have the following immediate corollary.

Corollary 5.4. *Given any parameter $0 < \varepsilon < 1$ and any planar graph P with n vertices and k terminals where $k \leq \sqrt{n}$ on $O(1)$ many faces, one can compute an ε-emulator of P with respect to the terminals of size $O(k \operatorname{poly}\log k/\operatorname{poly}\varepsilon)$ in time $O(n \operatorname{poly}\log^* n/\varepsilon^{O(1)})$ time.*

REFERENCES

[1] Alok Aggarwal, Maria M Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. 1987. Geometric applications of a matrix-searching algorithm. *Algorithmica* 2 (Nov. 1987), 195–208. https://doi.org/10.1007/BF01840359

[2] Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen Kobourov, and Richard Spence. 2020. Graph spanners: A tutorial review. *Computer Science Review* 37 (2020), 100253. https://doi.org/10.1016/j.cosrev.2020.100253

[3] A. Andoni, A. Gupta, and R. Krauthgamer. 2014. Towards $(1+\varepsilon)$-Approximate Flow Sparsifiers. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 279–293. https://doi.org/10.1137/1.9781611973402.20

[4] Yair Bartal and Lee-Ad Gottlieb. 2013. A Linear Time Approximation Scheme for Euclidean TSP. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, Berkeley, CA, USA, 698–706. https://doi.org/10.1109/FOCS.2013.80

[5] Here $\log^{(c)} n = \log(\ldots\log(\log n))$, where the logarithm is applied c times.

[5] A. Basu and A. Gupta. 2008. Steiner Point Removal in Graph Metrics. (2008). Unpublished manuscript, available from http://www.math.ucdavis.edu/~abasu/papers/SPR.pdf.

[6] T. Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. 2006. A Tight Lower Bound for the Steiner Point Removal Problem on Trees. In *9th International Workshop on Approximation, Randomization, and Combinatorial Optimization (Lecture Notes in Computer Science, Vol. 4110)*. Springer, 70–81. https://doi.org/10.1007/11830924_9

[7] Timothy M. Chan and Dimitrios Skrepetos. 2019. Faster Approximate Diameter and Distance Oracles in Planar Graphs. *Algorithmica* 81, 8 (Aug. 2019), 3075–3098. https://doi.org/10.1007/s00453-019-00570-z

[8] Hsien-Chih Chang, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2018. Near-Optimal Distance Emulator for Planar Graphs. In *26th Annual European Symposium on Algorithms (ESA 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 112)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 16:1–16:17. https://doi.org/10.4230/LIPIcs.ESA.2018.16

[9] Hsien-Chih Chang and Tim Ophelders. 2020. Planar Emulators for Monge Matrices. (2020), 7.

[10] Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. 2010. Vertex Sparsifiers and Abstract Rounding Algorithms. In *51st Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 265–274. https://doi.org/10.1109/FOCS.2010.32

[11] S. Chaudhuri, K. V. Subrahmanyam, F. Wagner, and C. D. Zaroliagis. 2000. Computing Mimicking Networks. *Algorithmica* 26 (2000), 31–49. Issue 1. https://doi.org/10.1007/s004539910003

[12] Yun Kuen Cheung. 2018. Steiner Point Removal: Distant Terminals Don't (Really) Bother. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*. SIAM, 1353–1360. https://doi.org/10.1137/1.9781611975031.89

[13] Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. 2016. Graph Minors for Preserving Terminal Distances Approximately – Lower and Upper Bounds. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*. 131:1–131:14. https://doi.org/10.4230/LIPIcs.ICALP.2016.131

[14] Julia Chuzhoy. 2012. On vertex sparsifiers with Steiner nodes. In *44th symposium on Theory of Computing*. ACM, 673–688. https://doi.org/10.1145/2213977.2214039

[15] Vincent Cohen-Addad, Andreas Feldmann, and David Saulpic. 2019. Near-Linear Time Approximations Schemes for Clustering in Doubling Metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. 540–559. https://doi.org/10.1109/FOCS.2019.00041

[16] M. Englert, A. Gupta, R. Krauthgamer, H. Räcke, I. Talgam-Cohen, and K. Talwar. 2014. Vertex Sparsifiers: New Results from Old Techniques. *SIAM J. Comput.* 43, 4 (2014), 1239–1262. https://doi.org/10.1137/130908440 arXiv:1006.4586

[17] Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. 2018. Holiest minimum-cost paths and flows in surface graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2018*. ACM Press, Los Angeles, CA, USA, 1319–1332. https://doi.org/10.1145/3188745.3188904

[18] Jeff. Erickson, Kyle. Fox, and Amir. Nayyeri. 2012. Global Minimum Cuts in Surface Embedded Graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1309–1318. https://doi.org/10.1137/1.9781611973099.103

[19] Jittat Fakcharoenphol and Satish Rao. 2006. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. System Sci.* 72, 5 (Aug. 2006), 868–889. https://doi.org/10.1016/j.jcss.2005.05.007

[20] Arnold Filtser. 2018. Steiner Point Removal with Distortion $O(\log k)$. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*. Society for Industrial and Applied Mathematics, 1361–1373. https://doi.org/10.1137/1.9781611975031.90

[21] Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. 2019. Relaxed Voronoi: A Simple Framework for Terminal-Clustering Problems. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, Vol. 69. 10:1–10:14. https://doi.org/10.4230/OASIcs.SOSA.2019.10

[22] Kyle Fox and Jiashuai Lu. 2020. A Near-Linear Time Approximation Scheme for Geometric Transportation with Arbitrary Supplies and Spread. (2020), 18.

[23] Greg N. Frederickson. 1987. Fast Algorithms for Shortest Paths in Planar Graphs, with Applications. *SIAM J. Comput.* 16, 6 (Dec. 1987), 1004–1022. https://doi.org/10.1137/0216064

[24] Gramoz Goranci, Monika Henzinger, and Pan Peng. 2020. Improved Guarantees for Vertex Sparsification in Planar Graphs. *SIAM Journal on Discrete Mathematics* 34, 1 (2020), 130–162. https://doi.org/10.1137/17M1163153

[25] Gramoz Goranci and Harald Räcke. 2016. Vertex Sparsification in Trees. In *Approximation and Online Algorithms - 14th International Workshop, WAOA*. 103–115. https://doi.org/10.1007/978-3-319-51741-4_9

[26] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2021. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2212–2228. https://doi.org/10.1137/1.9781611976465.132

[27] Anupam Gupta. 2001. Steiner points in tree metrics don't (really) help. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 220–227. http://dl.acm.org/citation.cfm?id=365411.365448

[28] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. 1998. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.* 57 (1998), 366–375. Issue 3. https://doi.org/10.1006/jcss.1998.1592

[29] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55, 1 (1997), 3–23. http://dx.doi.org/10.1006/jcss.1997.1493

[30] Alon Itai and Yossi Shiloach. 1979. Maximum flow in planar networks. *SIAM J. Comput.* 8, 2 (May 1979), 16.

[31] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2011. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing (STOC '11)*. San Jose, California, USA, 313–322. https://doi.org/10.1145/1993636.1993679

[32] Lior Kamma, Robert Krauthgamer, and Huy L. Nguyen. 2015. Cutting Corners Cheaply, or How to Remove Steiner Points. *SIAM J. Comput.* 44, 4 (2015), 975–995. https://doi.org/10.1137/140951382

[33] Nikolai Karpov, Marcin Pilipczuk, and Anna Zych-Pawlewicz. 2017. An Exponential Lower Bound for Cut Sparsifiers in Planar Graphs. *12th International Symposium on Parameterized and Exact Computation, IPEC* (2017), 24:1–24:11. https://doi.org/10.4230/LIPIcs.IPEC.2017.24

[34] Arindam Khan and Prasad Raghavendra. 2014. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.* 114, 7 (2014), 365–371. https://doi.org/10.1016/j.ipl.2014.02.011

[35] Philip N Klein. 2005. Multiple-source shortest paths in planar graphs. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. 146–155.

[36] Philip N. Klein, Shay Mozes, and Christian Sommer. 2013. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13*. ACM Press, Palo Alto, California, USA, 505. https://doi.org/10.1145/2488608.2488672

[37] P. N. Klein and S. Subramanian. 1998. A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs. *Algorithmica* 22, 3 (Nov. 1998), 235–249. https://doi.org/10.1007/PL00009223

[38] R. Krauthgamer, H. Nguyen, and T. Zondiner. 2014. Preserving Terminal Distances Using Minors. *SIAM Journal on Discrete Mathematics* 28, 1 (2014), 127–141. https://doi.org/10.1137/120888843

[39] Robert Krauthgamer and Havana (Inbal) Rika. 2020. Refined Vertex Sparsifiers of Planar Graphs. *SIAM Journal on Discrete Mathematics* 34, 1 (2020), 101–129. https://doi.org/10.1137/17M1151225

[40] Robert Krauthgamer and Inbal Rika. 2013. Mimicking Networks and Succinct Representations of Terminal Cuts. In *24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1789–1799. https://doi.org/10.1137/1.9781611973105.128

[41] Robert Krauthgamer and Tamar Zondiner. 2012. Preserving Terminal Distances Using Minors. In *39th International Colloquium on Automata, Languages, and Programming (Lecture Notes in Computer Science, Vol. 7391)*. Springer, 594–605. https://doi.org/10.1007/978-3-642-31594-7_50

[42] Konstantin Makarychev and Yury Makarychev. 2016. Metric extension operators, vertex sparsifiers and Lipschitz extendability. *Israel Journal of Mathematics* 212, 2 (2016), 913–959. https://doi.org/10.1007/s11856-016-1315-8

[43] Ankur Moitra. 2009. Approximation Algorithms for Multicommodity-Type Problems with Guarantees Independent of the Graph Size. In *50th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 3–12. https://doi.org/10.1109/FOCS.2009.28

[44] Shay Mozes, Cyril Nikolaev, Yahav Nussbaum, and Oren Weimann. 2018. Minimum cut of directed planar graphs in $O(n \log\log n)$ time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. New Orleans, Louisiana, 477–494. arXiv:1512.02068

[45] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* 7, 1 (March 1987), 105–113. https://doi.org/10.1007/BF02579206

[46] John H Reif. 1981. Minimum s-t Cut of a Planar Undirected Network in $O(n \log^2(n))$ Time. (1981), 12.

[47] R. Sharathkumar and Pankaj K. Agarwal. 2012. Algorithms for the Transportation Problem in Geometric Settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, Yuval Rabani (Ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, 306–317. https://doi.org/10.1137/1.9781611973099.29

[48] Mikkel Thorup. 2004. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM* 51, 6 (Nov. 2004), 993–1024. https://doi.org/10.1145/1039488.1039493

[49] Oren Weimann and Raphael Yuster. 2016. Approximating the Diameter of Planar Graphs in Near Linear Time. *ACM Transactions on Algorithms* 12, 1 (Feb. 2016), 1–13. https://doi.org/10.1145/2764910