Almost-Linear $\varepsilon$-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 $\varepsilon$-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 $\varepsilon$-emulators for planar graphs of almost-linear size $k^{1+o(1)}/\varepsilon$poly $\varepsilon$. In terms of $k$, this is a dramatic improvement over the previous quadratic upper bound of Cheung, Goranci and Henzingen [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:

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

$$\forall x, y \in T, \ dist_G(x, y) \leq dist_{G'}(x, y) \leq \alpha \cdot dist_G(x, y),$$

(1)

where $dist_G$ denotes the shortest-path distance in $G$ (and similarly for $G'$). In the important case that $\alpha = 1 + \varepsilon = e^{O(\varepsilon)}$ for $0 \leq \varepsilon \leq 1$, we simply say that $G'$ is an $\varepsilon$-emulator.† 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 $F$ for a fixed graph family $F$ (e.g., all planar graphs). If $F$ 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 $F$.

Vertex sparsifiers commonly exhibit a tradeoff between accuracy and size, which in our case of an emulator $G'$, are the distortion $\alpha$ and the number of vertices of $G'$. Let us briefly overview theknown 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 $O(k^3)$ [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 $\alpha$ [5, 6, 12, 21, 32]. For planar graphs, the known distortion bounds are $O(\log k)$ [20] and lower bound 2 [27].

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

*Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

STOC ’22, June 20–24, 2022, Rome, Italy.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9264-8/22/06...
https://doi.org/10.1145/3519935.3519998

‡Supported in part by NSF grant CCF-2006464.

Permission of the paper can be found on arXiv. The full version includes an improvement on the size of the emulator, from $k^{\Omega(1)}$poly $\varepsilon$ to $k^{1+o(1)}/\varepsilon$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 $O(poly \log n)$ 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.

1311
Our primary interest is in minimizing the size-bound when the distortion \( \alpha \) is \( 1 + \varepsilon \), i.e., \( \varepsilon \)-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 \( \tilde{O}(t^2/k^2) \) [13] and \( \Omega(k/t^2) \) [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 \( O(t) = t \cdot \text{polylog} t \), and multiplicative factors that depend on \( \varepsilon \) 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 \( \varepsilon \)-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 \( \varepsilon \)-emulator graph \( G' \) of size \( |V(G')| = k^{1+o(1)}/\text{poly} \varepsilon \). Furthermore, such an emulator can be computed deterministically in time \( O(n \log^\Omega(1) n/\text{poly} \varepsilon) \).

The result dramatically improves over the previous \( \tilde{O}(k^2/\varepsilon^2) \) 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>
<thead>
<tr>
<th>Distortion</th>
<th>Size (lower/upper)</th>
<th>Requirement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \Omega(k^2) )</td>
<td>planar</td>
<td>[9, 41]</td>
</tr>
<tr>
<td>1</td>
<td>( O(k^4) )</td>
<td>minor</td>
<td>[38]</td>
</tr>
<tr>
<td>1 + \varepsilon</td>
<td>( \Omega(k/\varepsilon) )</td>
<td>minor</td>
<td>[38]</td>
</tr>
<tr>
<td>1 + \varepsilon</td>
<td>( \tilde{O}(k^2/\varepsilon^2) )</td>
<td>minor</td>
<td>[13]</td>
</tr>
<tr>
<td>1 + \varepsilon</td>
<td>( k^{1+o(1)}/\text{poly} \varepsilon )</td>
<td>planar</td>
<td>Theorem 1.1</td>
</tr>
<tr>
<td>( O(\log k) )</td>
<td>( k )</td>
<td>minor</td>
<td>[20]</td>
</tr>
</tbody>
</table>

**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 \( \varepsilon \)-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 \( O(n \text{poly}(\log^* n)) \), 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 \( \varepsilon \)-MSSP data structure on \( G \) with respect to \( T \) in time \( O(n \text{poly}(\log^* n)/\text{poly} \varepsilon) \), 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 \( O(n \log n) \)-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 \( O(n \log^* n/\text{poly} \varepsilon) \).

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 \( O(n \log^2 n + \varepsilon^{-3} \log n) \)-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 \( O(n \log n \log^* n + n \log n/\text{poly} \varepsilon) \).

As the applications can be derived from applying the \( \varepsilon \)-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’s-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 lie 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 \( \varepsilon \)-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.

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 $1 + \epsilon$ and the final emulator size has polynomial dependency on $r^{-1}$. 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 $\Phi$ is the spread of the terminal distances (cf. Claim 4.4). When $\Phi \leq 2^{r^{0.3}}$, we will get the polynomially-small $O(r^{-0.1})$ 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 $\Phi > 2^{r^{0.3}}$, 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 $r^{0.3}$; 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 $\sim r^{-0.5}$-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. 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 $S \setminus S$, but now using $r$-covers $[48]$ for $r_r := r^{-0.1}$, and split along the paths to create sub-instances. Because the cluster is non-expanding and has moderate size, the number of terminals in $S \setminus S$ is at most $(r^{-0.7} - 1)|S| \leq r^{0.3}$, and thus the number of portals is $O(r^{0.3}/r_r) \leq O(r^{0.4})$, 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 recombinated instance to be an emulator. This can be done by observing that terminal pairs among $U \setminus S$ are far apart, and similarly when one terminal is from $S$ and the other is from $U \setminus S$; hence only portals involving $S \setminus S$ have to be dealt with using properties of $r_r$-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 $\hat{S}$ of lowest level that contains most of the terminals, and construct a collection of non-crossing shortest paths between terminals in $\hat{S}$ 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 $O(r^{0.7})$ levels from $\hat{S}$ because we cannot have nested expanding clusters for $O(r^{0.7})$ 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 $[36]$;

1313
(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 \(\epsilon\)-emulator that is sub-linear in the size of each piece in linear time (up to \(O_{\epsilon}(\log^* n)\) 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 in the graph). If we say that a Planar \(\epsilon\)-emulator preserves distances exactly, i.e., \(dist_G(x, y) = dist_G'(x, y)\) 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 \(\theta\)-emulator for \((G', T)\) of size \(|V(G')| \leq k^2\). 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 \((t_1, t'_2, t'_1, t_2)\) or \((t_1, t'_1, t'_2, t_2)\); 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 \(\mathcal{M}\) are non-crossing. Sometimes we abuse the language and say that a set of shortest paths \(P\) in \(G\) is **non-crossing** when the collection of endpoint pairs for the paths is non-crossing. The **Monge property** states that, for every one-hole instance \((G, T)\) and every crossing pairs of terminals \((t_1, t_2)\) and \((t'_1, t'_2)\),

\[
\text{dist}_G(t_1, t_2) + \text{dist}_G(t'_1, t'_2) \geq \text{dist}_G(t'_1, t_2) + \text{dist}_G(t_1, t'_2),
\]

**Well-structured sets of shortest paths.** Consider a graph \(G\) and a collection \(P\) of shortest paths in \(G\). We say that the set \(P\) is **well-structured** if for every pair of paths \((P, P')\) in \(P\), \(P \cap P'\) 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 \(P\) of shortest paths, one for each pair of terminals in \(T\) in \(O(|E(G)| \cdot \log |\mathcal{M}|)\) time.

---

**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.

\( \varepsilon \)-cover. We use the notion of \( \varepsilon \)-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 \( \varepsilon \)-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 \( \text{dist}_G(v, y) + \text{dist}_G(y, x) \leq (1 + \varepsilon) \cdot \text{dist}_G(v, x) \). Small \( \varepsilon \)-cover of size \( O(1/\varepsilon) \) is known to exist.

**Theorem 3.1** (Thorup \([48, \text{Lemma 3.4}]\)). Let \( \varepsilon \in (0, 1) \) be a constant. For every shortest path \( P \) in some graph \( G \) and every vertex \( v \not\in P \), there is an \( \varepsilon \)-cover of \( v \) on \( P \) with size \( O(1/\varepsilon) \). Moreover, such an \( \varepsilon \)-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 \([37, \text{Lemma 4}] \) 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 \( \varepsilon \)-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 \( G \) be 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 \( \varepsilon \)-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 \( \varepsilon \)-emulators for one-hole instances, as stated in Theorem 3.1. We say that an \( \varepsilon \)-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 \( \varepsilon \)-emulator for \((G, T)\) of size \( k \log^{O(1)} k / \text{poly} \varepsilon \) in \( O((n + k^2) \log^{O(1)} n / \text{poly} \varepsilon) \) 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 \( \lambda := c^4 \log^2 k / \varepsilon^2 \), where \( k := |T| \) and \( c^4 > 0 \) 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 a one-hole instance \((H, U)\) with \( r := |U| \) and threshold \( \lambda := c^4 \log^2 k / \varepsilon^2 \), one can compute a collection of one-hole instances \(\{(H_1, U_1), \ldots, (H_n, U_n)\}\), such that

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

Moreover, given an \( \varepsilon \)-emulator \((H'_i, U'_i)\) for each \((H_i, U_i)\), algorithm **Combine** computes for \((H, U)\) an \((\varepsilon + O(\log^{O(1)} r))\)-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 \( O(|V(H)| + r^2 \cdot \log r \cdot \log |V(H)|) \).

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 \( \tau \). 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 \((H_1, U_1), \ldots, (H_n, U_n)\) generated by Lemma 3.2. The leaves of \( \tau \) 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 \( \tau \) inductively in a bottom-up fashion: Given a non-leaf node \((H, U)\) with children \((H_1, U_1), \ldots, (H_n, U_n)\), 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 \((H'_i, U'_i), \ldots, (H'_n, U'_n)\) to obtain an emulator \((H', U)\) for instance \((H, U)\). After all nodes in \( \tau \) 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 \( k \log^{O(1)} k / \text{poly} \varepsilon \). In what follows, \( \mathcal{H} \) denotes this set at the end of the first stage.

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

\[^3\text{This step can use any 0-emulator that has size poly } k \text{ and can be constructed in time } O(n + \text{poly} k), \text{ and we conveniently use Theorem 2.1.} \]
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
\[ |V(G')| \leq \sum_{(H,U) \in \mathcal{H}} |U|^2 + \sum_{(H,U) \in \mathcal{E}(r)} f(H,U). \]

- We bound the term \( \sum_{(H,U) \in \mathcal{E}(r)} f(H,U) \) via a charging scheme as follows. Consider a node \((H,U)\) in \(r\). We charge the value of \(f(H,U)\) uniformly to all terminals in \(U\), so each terminal has a charge of at most \(\hat{c} \log^2 k\). The height of \(r\) 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 \(\bigcup(H,U) \in \mathcal{U}\) 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
\[
\sum_{(H,U) \in \mathcal{H}} |U|^2 \leq \max_{(H,U) \in \mathcal{H}} \{|U|\} \cdot \sum_{(H,U) \in \mathcal{H}} |U| \\
\leq \lambda \cdot k \log^O(1) k = k \log^O(1) k / \poly \epsilon.
\]

Therefore, \(|V(G')|\) is at most
\[
\sum_{(H,U) \in \mathcal{H}} (|U|^2 + |U| \cdot O(\log^2 k)) = k \cdot \log^O(1) k / \poly \epsilon.
\]

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 \(r\) with children \((H_1,U_1), \ldots, (H_s,U_s)\).

- For instances \((H_i,U_i)\) with \(|U_i| \leq \lambda\) (which will all be in \(\mathcal{H}\) at the end of first stage), charge every vertex in \(U_i\) to vertices in \(U\). Since \(\sum_{|U_i| \leq \lambda} |U_i| \leq O(\log^2 |U| \cdot |U|) \leq O(\log^2 k \cdot |U|),\) 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 \(|U'| \leq O(|U|/\log^2 k)|U|\) by Lemma 3.2. Charge every vertex in \(U'\) uniformly to vertices in \(U\), so each vertex gets \(O(1/\log^2 k)\) charge. We call these charge active.

The total inactive charge on each vertex of \(T\) is \(O(\log^3 k)\) because \(r\) 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 \(O(\log(1/\log(10/9) \lambda - 1)) \leq 1/2\). Note that this only accounts for the direct active charge. For example, some terminal does not belong 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 \(1/2 + (1/2)^2 + \cdots \leq 1\).

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 \(k \log^O(1) k / \poly \epsilon\).

Correctness. It remains to show that \((G',T)\) is an \(\epsilon\)-emulator for \((G,T)\). Recall that we have associated with the algorithm in first stage a (rooted) partitioning tree \(\tau\). We now define, for each tree node \((H,U)\), a value \(\epsilon(H,U)\) as follows. If \((H,U)\) is a leaf node, we define \(\epsilon(H,U) := 0\). Otherwise, \((H,U)\) is a non-leaf node with child nodes in \(r\) 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 \(c < (\epsilon^*)^{1/\poly\epsilon}\). We define
\[
\epsilon(H,U) := \frac{c \log^4 r}{\rho \epsilon} + \max\{\epsilon(H_1,U_1), \ldots, \epsilon(H_s,U_s)\}.
\]

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

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

\section{Recursive Emulator Construction}

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

\subsection{Splitting and Gluing}

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

\textbf{Splitting}. The input to procedure \textsc{Split} consists of

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

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

1316

Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan
Consider the plane graph $H$ with all the terminals in $U$ lying on the outerface of $H$. Because all the paths in $P$ 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 $H$ 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 $P$. 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 $\tilde{Z}$ the resulting graph. We then add to $\tilde{Z}$ all the branch vertices of $P$ vertices that are not in $Y$ (and thus have not been merged), where the branch vertices of $P$ in $H$ are those vertices that has degree at least three in the union of shortest paths in $P$. 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 clock-wise order as $(v_1, \ldots, v_m)$. 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 dist$_H(v_i, v_{i-1})$; similarly, add an edge connecting $v_i$ to $v_{i+1}$ with edge-weight 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 $\tilde{Z}$. Graph $Z$ is naturally a plane graph by inheriting the embeddings of all $Z_R$s. By the assumption that $Y$ contains all the endpoints of paths in $P$, 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 $((H, U), P, Y)$ 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

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

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

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 $(H, U)$ 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

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

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

**4.2 Small Spread Case**

In this case we assume $\Phi \leq 2^{\Theta(1)} \log^2 r$. 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

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

Balanced terminal pairs. Let $(H, U)$ be a one-hole instance with $U = \{u_1, \ldots, u_k\}$, 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 $j - i \leq c \cdot r$ and $i + r - j \leq c \cdot r$. 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 $Y$ contain the endpoints of $P$, together with the following vertices of $P$: for each $1 \leq i \leq L_r$,

(i) among all vertices $v$ of $P$ with $\text{dist}_P(v, u) \leq e^i \varepsilon r$, vertex that maximizes its distance to $u$;
(ii) among all vertices $v$ of $P$ with $\text{dist}_P(v, u) \geq e^i \varepsilon r$, vertex that minimizes its distance to $u$;
(iii) among all vertices $v$ of $P$ with $\text{dist}_P(v, u') \geq e^i \varepsilon r$, vertex that maximizes its distance to $u'$;
(iv) among all vertices $v$ of $P$ with $\text{dist}_P(v, u') \geq e^i \varepsilon r$, vertex that minimizes its distance to $u'$.

In other words, if we think of path $P$ as a line, and then mark, for each $1 \leq j \leq L_r$, 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 $u'$, then set $Y$ contains, for all marked points, the vertices of $P$ that are closest to it from both sides. By definition, $|Y| \leq 4L_r$.

We apply the procedure $\text{Split}$ 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 $\text{Split}$, 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 $|Y| \leq 4L_r = r/25 \log^2 r$, we get that $|U_1| \leq (3/4)r + r/25 \log^2 r \leq (9/10)r$, and similarly $|U_2| \leq (9/10)r$. Also, $|U_1| + |U_2| \leq |U| + O(|Y|) \leq r \cdot (1 + O(L_r/r)) = r \cdot (1 + O(1/\log r))$.

We now construct an algorithm $\text{Combine}$ that satisfies the required properties. Let $(H'_1, U'_1)$ be an $e$-emulator for $(H_1, U_1)$ and let $(H'_2, U'_2)$ be an $e$-emulator for $(H_2, U_2)$. The algorithm $\text{Combine}$ simply applies the procedure $\text{Glue}$ 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 $\text{Combine}$ then returns the instance $(H', U')$. It remains to show that the algorithm $\text{Combine}$ satisfies the required properties. Note that instance pair $(H'_1, U'_1), (H'_2, U'_2)$ is also a valid input for procedure $\text{Glue}$. 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 $(\hat{H}, \hat{U})$ 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 $e$-emulator for $(\hat{H}, \hat{U})$. From Claim 4.4, instance $(\hat{H}, \hat{U})$ is a $(3\varepsilon r)$-emulator for instance $(H, U)$. Altogether, $(H', U)$ is an $(e + 3\varepsilon r) = (\varepsilon + O(\frac{\log^2 r}{r^2}))$-emulator for $(H, U)$.

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

From the procedure $\text{Split}$, $\hat{H}_1$ 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 $\hat{H}_2$ 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 $\hat{H}_1$ and $\hat{H}_2$. 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 $\tilde{H}$, $H_1$, and $H_2$. Left: Graphs $H_1$ (top) graph $H_2$ (bottom) viewed as individual graphs. Right: Subgraph $\tilde{H}$ obtained by gluing graphs $H_1$ and $H_2$. Vertices in $Y \setminus \{u, u’\}$ are shown in purple.](image)

We first show that for each pair $u_1, u_2$ of terminals in $U$ one has $\text{dist}_{\hat{H}}(u_1, u_2) \leq \text{dist}_{\hat{H}}(u_1, u_2)$. 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 $P$ that lies entirely in $\hat{H}_1$. From the construction of $\hat{H}$, the same path belongs to graph $H_1$, and therefore $\text{dist}_{\hat{H}}(u_1, u_2) \leq \text{dist}_{\hat{H}}(u_1, u_2)$. 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 $P^2$ 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 $\tilde{H}$. Therefore, $\text{dist}_{\hat{H}}(u_1, u_2) \leq \text{dist}_{\hat{H}}(u_1, u_2)$.

From now on we focus on showing that, for each pair $u_1, u_2$ of terminals in $U$, $\text{dist}_{\hat{H}}(u_1, u_2) \leq e^{3\varepsilon r} \cdot \text{dist}_H(u_1, u_2)$. 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 $\text{dist}_{\hat{H}}(u_1, u_2) = \text{dist}_{\hat{H}}(u_1, u_2)$. 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). Let $\tilde{Q}$ be the $u_1-u_2$ shortest path in
We consider the following possibilities, depending on the relative values of $Y$ without loss of generality that $|x_1 - x_2| = 0$ and $|u_1 - u_2| = 0$. Assume not, then the path $P$ connects $x_1$ to $x_2$ and $u_1$ to $u_2$. So the distance between $x_1$ and $u_2$ is less than the distance between $x_1$ and $x_2$. However, because $|U|/2 \leq |V(H)| \leq (3/4) \cdot |U|$, at least one of the pairs $(u_1, u_2)$ 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 $1 \leq j \leq L$, the point on the line that is at distance $e^{cr^j}$ from $u$, and the point on the line that is at distance $e^{cr^j}$ from $u'$; these marked points are called landmarks. Observe that there is no landmark between vertices $x_1$ and $x_2$: If there is a landmark between vertices $x_1$ and $x_2$, then set $Y$ contains, for all landmarks, 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 $x_2$ and are added to vertex set $Y$, a contradiction. Let $x$ be the landmark closest to $x_1$ that lies between $u$ and $x_2$, and assume $\delta_P(x, u) = e^{cr^j}$. Let $y$ be the vertex of $Y$ closest to the landmark $x$ that lies between $x$ and $x_1$. From the construction of terminals, $e^{cr^j} \leq \delta_P(y, u) < \delta_P(x_1, u), \delta_P(x_2, u) < e^{(1+\delta^j)}r^j$. Therefore, $\delta_P(x_1, y), \delta_P(x_2, y) < (e^{cr^j} - 1) \cdot e^{cr^j}$. Consider now the $u_1-u_2$ path in $H$ formed by concatenation of (i) the copy of $Q_1$ in $H_1$ connecting $u_1$ to the copy of $x_1$ of $H_1$; (ii) the subpath of $P$ connecting $x_1$ to $y$; (iii) the subpath of $P$ connecting $y$ to the copy $x_2$ of $x_2$; and (iv) the copy of $Q_2$ in $H_2$ connecting $x_2$ to $u_2$. Calculating the weight of this path is at most $e^{3cr^j} \cdot \delta_P(x_1, u_1)^2$. Therefore, $\delta_P(x_1, u_1) \leq e^{3cr^j} \cdot \delta_P(u_1, u_2)$. This completes the proof of Claim 4.4.

### 4.3 Large Spread Case

In this case we assume $\Phi > 3^{3\theta \log^2 r}$. Without loss of generality $\min_{u \in U} \delta_G(u, u') = 1$, so $\max_{u, u' \in U} \delta_G(u, u') = \Phi$. In the algorithm for this case, we use the following parameters:

$$
\mu := r^2, \quad L := \lceil \log_\mu \Phi \rceil, \quad c_r := \frac{\log r}{\mu^{1/3}}, \quad \epsilon_r := \frac{1}{r^{\mu^{1/3}}}
$$

We first compute a hierarchical partitioning $(S_0, S_1, \ldots, S_L)$ of terminals in $U$ in a bottom-up fashion as follows. We proceed in $L$ iterations. In the $i$th iteration, we compute a collection $S_i$ of subsets of $U$ that partition $U$.

We start by letting collection $S_0$ contain, for each terminal $u \in U$, a singleton set $\{u\}$. That is, $S_0 = \{\{u\} \mid u \in U\}$. Assume we have already computed the collection $S_{i-1}$ of subsets. We now describe the construction of collection $S_i$, as follows. First, let graph $W_{i-1}$ be obtained from $H$ by contracting each subset $S \in S_{i-1}$ into a single supernode, that we denote by $v_S$, and we define $V_{i-1} = \{v_S \mid S \in S_{i-1}\}$. 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 $\delta_{W_{i-1}}(v_S, v_{S'}) \leq \mu^i$, or equivalently $\delta_{P}^{\prime}(S, S') \leq \mu^i$. Finally, we define $S_i$ to be the collection that contains, for each connected component $C$ of graph $R_{i-1}$, the set $\cup_{v \in V(C)} S$. It is easy to verify that the sets in $S_i$ partition $U$. This completes the description of the hierarchical partitioning $(S_0, S_1, \ldots, S_L)$. Clearly, collection $S_L$ contains a single set $U$. We denote $S = \cup_{0 \leq i \leq L} S_i$. So collection $S$ is a laminar family. That is, for every pair $S, S' \in S$, 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 $S \in S_i$, $\delta_G(S) \leq 2r \cdot \mu^j$.

It would be convenient for us to associate a partitioning tree $\tau$ with the hierarchical partitioning $(S_0, S_1, \ldots, S_L)$ we have computed, in a natural way as follows. Its vertex set $V(\tau)$ is $V_0 \cup \ldots \cup V_L$ (recall that for each $i$, $V_i = \{v_S \mid S \in S_i\}$, that is, $V_i$ contains, for each set $S \in S_i$, the supernode $v_S$ representing $S$). We call nodes in $V_i$ level-$i$ nodes of tree $\tau$, and we call sets in $S_i$ level-$i$ sets. Since $S_L = \{U\}$, there is only one level-$L$ node in $\tau$, that we view as the root of $\tau$. The edge set $E(\tau)$ contains, for each pair $S, S'$ of sets such that $S \subseteq S_i, \hat{S} \subseteq S_{i+1}$ for some $i$ and $S \subseteq \hat{S} \subseteq S_i$ an edge connecting $v_S$ to $v_{S'}$, so $v_S$ is a child node of $v_{S'}$, 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 $\tau$ is indeed a tree.

### Observation 4.7. Let $S, S'$ be distinct sets in $S$ 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 $\delta_{G}(u_3, u_4) \leq \mu^i$, $\delta_{G}(u'_3, u'_4) \leq \mu^{i'}$ 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 $y_1$ the boundary segment clockwise from $u'_3$ to $u'_4$ around the outerface of $H$, and denote by $y_2$ the boundary segment clockwise from $u'_3$ to $u'_4$ around the outerface of $H$, then since we have assumed that $(u_3, u_4)$ and $(u'_3, u'_4)$ 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 $R_1, \ldots, R_{t-1}$ and collections $𝒮_1, \ldots, 𝒮_t$, 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 $0 \leq j \leq t$, $\text{dist}(u^j, u^{j+1}) \leq \rho_i^j$; and for every pair $u, u'$ of terminals do not belong to the same level-$i$ set, $\text{dist}(u, u') > \rho_i^j$.

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 $\rho_i^j$. 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 $(u'_3, u'_4)$ are crossing and $\text{dist}(u_3, u_4) \leq \rho_i^j$.

We can then use similar arguments to find another pair $(u'_3, u'_4)$ crossing the pair $(u_3, u_4)$ and $\text{dist}(u'_3, u'_4) \leq \rho_i^j$. Note that, since $u_3, u_4 \in S$ and $u'_3, u'_4 \notin S$, $\text{dist}(u_3, u_4) > \rho_i^j$ and $\text{dist}(u'_3, u'_4) > \rho_i^j$. Altogether, we get that

$$\text{dist}(u'_3, u'_4) + \text{dist}(u_3, u_4) \leq \rho_i^j + \rho_i^j \leq \rho_i^j + \rho_i^j < \text{dist}(u_3, u_4) + \text{dist}(u_3, u_4),$$

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 $r^*_i = \frac{1}{2\rho_i}$. We say that a set $S \in \mathcal{S}$ is expanding, if and only if $|S| \geq e^{r^*_i} \cdot |S|$, where $\hat{S}$ is the parent set of $S$ (or equivalently $\hat{S}_q$ is the parent node of $S_q$ in $r$), otherwise we say it is non-expanding. We now distinguish between two cases, depending on whether or not $S$ contains a non-expanding set with moderate size.

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

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 $e_r$, and compute an $e_r$-cover of $u'$ on $P_{(u, u')}$. We then let $Y_{(u, u')}$ be the union of all such $e_r$-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 $V(P)$ that contains all endpoints of paths in $\mathcal{P}$. Moreover, from Lemma 2.4,

$$|Y \setminus U| \leq O\left(\frac{|S'|}{e_r}\right) \leq O\left(\frac{e_{r^*_i} - 1 - r}{e_r}\right) = O\left(\frac{\rho_i^j}{\log r}\right).$$

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 $r/6 \leq |S|, |S'| \leq 5r/6$ and $|Y| \leq O(r\log^2 r)$. On the one hand, each instance in $\mathcal{H}_2$ contains at most two terminals in $S$, and so it contains at most $r - |S| + 2 + |Y| \leq (9/10)r$ terminals. On the other hand, each instance in $\mathcal{H}_2$ does not contain terminals in $S'$, and so it contains at most $r - |S'| + |Y| \leq (9/10)r$ terminals.

Second, note that $|Y| \leq O\left(\frac{\rho_i^j}{\log r}\right)$ and $\lambda = e^{-2} \cdot k / \Delta_2$, then from Claim 4.2 (by setting $\mu = \lambda$), we get that $\sum_{(H, U) \in \mathcal{H}} |U| \leq O(r \log^2 r)$ and $\sum_{(H, U) \in \mathcal{H}} |U| \leq |U| \cdot (1 + O\left(\frac{1}{\log r}\right))$.

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 $\epsilon$-emulator for $(H_i, U_i)$. The algorithm Combine simply applies the procedure Glue to $(H'_i, U_i), \ldots, (H'_i, U_i)$ 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 $(H_i, U_i), \ldots, (H_i, U_i)$ is also a valid input for procedure Glue. Let $(H, U)$ be the one-hole instance that the procedure Glue outputs when it is applied to instances $(H_i, U_i), \ldots, (H_i, U_i)$. It is easy to verify that $U = U$. We use the following claim whose proof is in the full version.

Claim 4.8. Instance $(H, U)$ is an $(\epsilon_r)$-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 $1 \leq i \leq t$, $(H'_i, U_i)$ is an $\epsilon$-emulator for $(H_i, U_i)$, from Claim 4.3, $(H', U)$ is an $\epsilon$-emulator for $(H, U)$. Then from Claim 4.4 and Claim 4.8, we get that $(H', U)$ is an $(\epsilon + O(\epsilon_r))$-emulator for $(H, U)$. Moreover, from the algorithm Glue, the number of vertices in graph $H'$ that does not belong to graphs $H'_1, \ldots, H'_t$ 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, $|V(H')| \leq \left(\sum_{1 \leq i \leq s} |V(H'_i)|\right) + O(\log^2 r \cdot |U|)$.}

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 $\tilde{S}$ is heavy. Clearly, every level of $\tilde{v}$ contains at most one heavy node, and the set of heavy nodes induce a path in $r$ which ends at the root node of $\tilde{r}$. Let $\tilde{S}$ be the non-expanding heavy set that lies on the lowest level. We denote by $\tilde{L}$ the level that $\tilde{S}$ lies in and let $\tilde{S}$ be its parent set. Define $S^t = \tilde{S} \setminus \tilde{S}$ and $S^t = U \setminus S$. So sets $S^t, \tilde{S}, S'$ partition set $U$, and $|S^t| \leq (e^{r'} - 1) r$. We perform the same operations as in the Balanced Case (Section 4.3.1) to graph $H$ with respect to the partition $(\tilde{S}, S', S')$. Let $H'$ be the collection of instances we obtain. From similar analysis as in Section 4.3.1, we get that $\sum_{(H_i, U_i) \in H'} |U_i| \leq O(r \log^2 r)$, and $\sum_{(H_i, U_i) \in H'} |U_i| \leq r \cdot (1 + O(\frac{1}{\log r}))$. If additionally we have, for each $(H_i, U_i) \in H'$, $|U_i| \leq 0.9 r$, then we simply return the collection $H'$ as the output. Assume now that there exists some instance $(H', U') \in H'$ with $|U'| > 0.9 r$. Since $\sum_{(H_i, U_i) \in H'} |U_i| \leq r \cdot (1 + O(\frac{1}{\log r}))$, we may have only one such instance. It is easy to see from the algorithm Split that no terminal of $U'$ is a cut vertex in graph $H'$. Note that it is now enough to prove Lemma 3.2 for the instance $(H', U')$, which we do in the second step. Indeed, if Lemma 3.2 holds for instance $(H', U')$, then we simply apply the algorithm from Lemma 3.2 to instance $(H', U')$ and obtain a collection $H''$ instances. We simply return the collection $H' = (H' \cup (H^*, U^*)) \cup H''$. It is easy to verify that the output collection $H'$ 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 $(H', U')$ into instances with bounded spread, and eventually apply the algorithm from Case 1 to them. Consider the instance $(H', U')$. From the algorithm Split, the instance $(H', U')$ corresponds to a region of graph $H$, which is surrounded by shortest paths connecting terminals in $U$. Therefore, for every pair $v, v'$ of vertices in $H'$ (that are also vertices in $H$), dist$_H(v, v') = $ dist$_G(v, v')$. Note that set $U'$ can be partitioned into two subsets: set $\tilde{S}$ contains all terminals in $\tilde{S}$ that lies in $U'$, and set $Y$ contains all new terminals (which are vertices in $e^r$-covers of vertices of $S^t$ on paths of $P$) added in Step 1 that lie on the boundary of graph $H'$. Note that the distances between a pair of terminals in $Y'$ and the distances between a terminal in $Y'$ and a terminal in $\tilde{S}$ could be very small (even much smaller than $\min_{w, u'}$ dist$_H(u, u')$) at the moment, which makes it hard to bound the spread from above. Therefore, we start by modifying the instance $(H', U')$ as follows.

We let graph $H'$ be obtained from $H'$ by adding, for each terminal $u \in Y'$, a new vertex $\tilde{u}$ and an edge $(u, \tilde{u})$ with weight $\frac{1}{2^r}$. We then define $U' = \tilde{S} \cup \{\tilde{u} | u \in Y\}$. This completes the construction of the new instance $(H', U')$. We call this operation **terminal pulling**, it is easy to verify that $(H', U')$ is a one-hole instance, and moreover, for each new terminal in $U' \setminus \tilde{S}$, the distance in $H'$ from it to any other terminal in $\tilde{U}$ is at least $\frac{1}{2^r}$-1. We denote $\tilde{Y} = \{\tilde{u} | u \in Y\}$, 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 $(H', U')$. We now construct the hierarchical clustering $\tilde{S}$ for instance $(H', \tilde{U})$, in the same way as the hierarchical clustering $\tilde{S}$ for instance $(H, U)$, described at the beginning of the large spread case. Let $\tilde{r}$ be the partitioning tree associated with $\tilde{S}$. Recall that for every pair of vertices in $H'$, the distance between them in $H'$ is identical to the distance between them in $H$. From the construction of instance $(H', \tilde{U})$, it is easy to verify that both $\tilde{S}$ and $\tilde{r}$ has depth $\tilde{L}$, and in levels $L - 1, \ldots, 1$, new terminals in $\tilde{U} \setminus \tilde{S}$ only form singleton sets as each of them is at distance at least $\mu^{L-1}$ from any other terminal in $\tilde{U}$. Therefore, every non-singleton set in $\tilde{S}$ is also a set in $\tilde{S}$. We say that a set is **good** if

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

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

**Observation 4.9.** All good sets in $\tilde{S}_g$ lie on level at least $\tilde{L} - 10 \log r / e^r$. Every terminal either forms a singleton set on level at least $\tilde{L} - 10 \log r / e^r$, or belongs to some good set in $\tilde{S}_g$. Now for each good set $S$, we compute its border path set $\tilde{P}_S$ in instance $(H, \tilde{U})$ in the same way as in the Balanced Case (Section 4.3.1). Now define $\tilde{P} = \bigcup_{S \in \tilde{S}_g} \tilde{P}_S$. We show in the next observation that the collection $\tilde{P}$ of paths is non-crossing.

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

**Proof:** Assume for contradiction that the collection $\tilde{P}$ of paths is not non-crossing. Then there exist two distinct sets $S, S' \subseteq \tilde{S}_g$, 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 $S \subseteq \tilde{S}_g$. We define $S^* = \tilde{S} \setminus S$, where $\tilde{S}$ is the parent set of $S$ in $\tilde{S}_g$. Recall that a pair $(u, u')$ of terminals in $S$ is a border pair, if and only if the outer-boundary of $H$ connecting $u$ to $u'$ contains no other vertices of $S$ but at least one vertex that does not lie in $\tilde{S}$. Now for each border pair $(u, u')$, let $P_{(u,u')}^{(u,u')} = (u,u')$ be the $u-u'$ shortest path in $\tilde{P}_S$ 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 $e_r$, and compute an $e_r$-cover of $u^*$ on $P_{(u,u')}^{(u,u')}$. We then let $Y_{(u,u')}$ be the union of all such $e_r$-covers and the endpoints of $P_{(u,u')}^{(u,u')}$. We then set $Y_0 = \bigcup_{(u,u') \subseteq S^*} Y_{(u,u')}$ for a border pair $(u, u')$. Finally, we set $Y = \bigcup_{S \subseteq \tilde{S}_g} Y_S$, so $Y$ is a vertex set of $V(\tilde{P})$ that contains all endpoints of paths in $\tilde{P}$. Moreover, from Lemma 2.4,

$$|Y| \leq O \left( \sum_{S \subseteq \tilde{S}_g} |S| \right) \leq O \left( \frac{\left(e^{r'} - 1\right) \cdot \sum_{S \subseteq \tilde{S}_g} |S|}{e_r} \right) \leq O \left( \frac{\left(e^{r'} - 1\right) \cdot r}{e_r} \cdot \frac{1}{\log^2 r / r^{0.1}} \right) = O \left( \frac{r^{0.4}}{\log^2 r} \right)$$
We now apply the algorithm SPLITT to instance \((\hat{H}, \hat{U})\), the path set \(\hat{P}\) and the vertex set \(Y\). Let \(\mathcal{H}\) be the collection of one-hole instances we get. If all instances \((\hat{H}, \hat{U})\) in \(\mathcal{H}\) satisfy that \(|\hat{U}| \leq 0.9r\), then we terminate the algorithm and return \(\mathcal{H}\). Assume that there is some instance \((\hat{H}, \hat{U})\) in \(\mathcal{H}\) 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 SPLITT, \((\hat{H}, \hat{U})\) corresponds to a region of \(\hat{H}\) surrounded by segments of paths in \(\hat{P}\). 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 \(\hat{P}\), then from the definition of border pairs and border path sets, \(\hat{U}\) may not contain terminals in any good set of \(\tilde{S}_g\) other than \(\mathcal{H}\). However, since \(|S| \leq r/5\) and \(|Y| \leq O\left(\frac{\mu^2 \log^2 r}{r^2}\right)\), this contradicts to the assumption that \(|\hat{U}| > 0.9r\). Therefore, for each set \(S \in \tilde{S}_g\), the boundary of \(\hat{H}\) contains at most one segment of some boundary path of \(\mathcal{P}_S\).

We now modify the instance \((\hat{H}, \hat{U})\) as follows. Denote \(L^* = \hat{L} - 10 \log r/r^\epsilon\). Let \(H^*\) be the graph obtained from \(\hat{H}\) by applying the terminal pulling operation to every terminal in \(\hat{U} \setminus \hat{S}\) via an edge of weight \(\mu^2 \log^2 r\). We then define set \(U^*\) to be the union of \((\hat{U} \cap \hat{S})\) and the set of all new terminals created in the terminal pulling operation. We use the following observation.

**Observation 4.11.** \(\Phi(H^*, U^*) \leq \alpha \log^2 r/\epsilon r^{\epsilon} \).

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

Since \(\beta \log^2 r/\epsilon r^{\epsilon}\) is \(\alpha \log^2 r/\epsilon r^{\epsilon}\) in some large enough constant, we apply the algorithm from Case 1 to \((H^*, U^*)\) and obtain a collection of instances \(\mathcal{H}(H, \hat{U})\). The output of the algorithm is the collection \(\{\mathcal{H}(\hat{H}, \hat{U})\} \cup \mathcal{H}(H, \hat{U})\).

**Analysis of Unbalanced Case.** Recall that in this step we must assume that, after Step 1, there is an instance \((H_0, U_0)\) with \(|U_0| > 0.9r\), and we transformed it into another instance \((\hat{H}, \hat{U})\). We first show that it is sufficient to prove Lemma 3.2 for instance \((H, \hat{U})\). All other conditions can be easily verified. We now show that when applying the algorithm GLUE to \(\epsilon\)-emulators \(\{(H_0', U_0')\}_{i \in \mathbb{S}}\), we still obtain an \(\epsilon + O\left(\frac{\log r}{\log \mu}\right)\)-emulator for \((H, \hat{U})\).

Now we consider the case where one or both of the \((H_i, U_i)\) instances are not disjoint from \(\mathcal{S}_g\). Denote \(\tilde{S}_g\) to be the union of all \(S_i \cap \mathcal{S}_g\). Note that \(\tilde{S}_g\) is a subset of \(\mathcal{S}_g\), and that \(\tilde{S}_g\) is different from \(\mathcal{S}_g\) when \(\tilde{S}_g\) is not a subset of \(\mathcal{S}_g\). Therefore, we can divide the \(\tilde{S}_g\) instances into two cases: (1) \(\tilde{S}_g\) is a subset of \(\mathcal{S}_g\), and (2) \(\tilde{S}_g\) is not a subset of \(\mathcal{S}_g\).

**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 \(\epsilon\)-emulator for any \(O(1)\)-hole instance of size \(O(k \log k)\) can be computed in \(O(n \log(n)^2)\) 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
Almost-Linear $\epsilon$-Emulators for Planar Graphs

Theorem 5.3. Given any parameter $0 < \epsilon < 1$ and any planar graph $P$ with $n$ vertices and $k$ terminals where $k \leq \sqrt{n}$, let $\epsilon$-emulator such that

$$O\left(\frac{n}{\left(\log n\right)^{C_1}} \cdot \left(\log n\right)^C\right)$$

has size $O_{\epsilon}(n/\left(\log n\right)^{C_1})$, where $C_i = \log \epsilon$. The new graph $P'$ has size $O_{\epsilon}(k \polylog k)$.

Finally, consider the following immediate corollary.

Corollary 5.4. Given any parameter $0 < \epsilon < 1$ 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 $\epsilon$-emulator of $P$ with respect to the terminals of size $O(k \polylog n/\epsilon)$ in time $O(n \polylog n/\epsilon^{O(1)})$ time.

REFERENCES


Here $\log n^{O(1)} = \log(\ldots \log(n))$, where the logarithm is applied $c$ times.