Near-Optimal $(1+\epsilon)$-Approximate Fully-Dynamic All-Pairs Shortest Paths in Planar Graphs

Arnold Filtser
Computer Science Department
Bar Ilan University
Ramat Gan, Israel
arnold.filtser@biu.ac.il

Gramoz Goranci
Faculty of Computer Science
University of Vienna
Vienna, Austria
gramoz.goranci@univie.ac.at

Neel Patel
Department of Computer Science
University of Southern California
Los Angeles, USA
neelbpat@usc.edu

Maximilian Probst Gutenberg
Department of Computer Science
ETH Zurich
Zurich, Switzerland
maximilian.probst@inf.ethz.ch

***Abstract*—We study the fully-dynamic all-pair shortest paths (APSP) problem on planar graphs: given an n-vertex planar graph $G=(V,E)$ undergoing edge insertions and deletions, the goal is to efficiently process these updates and support distance and shortest path queries. We give a $(1+\epsilon)$-approximate dynamic algorithm that supports edge updates and distance queries in $n^{o(1)}$ time, for any $1/\text{poly}(\log n) < \epsilon < 1$. Our result is a significant improvement over the best previously known bound of $\tilde{O}(\sqrt{n})$ on update and query time due to [Abraham, Chechik, and Gavoille, STOC '12], and bypasses a $\Omega(\sqrt{n})$ conditional lower-bound on update and query time for exact fully dynamic planar APSP [Abboud and Dahlgaard, FOCS '16]. The main technical contribution behind our result is to dynamize the planar emulator construction due to [Chang, Krauthgamer, Tan, STOC '22].**

***Index Terms*—Planar Graph, Distance Oracles, Dynamic Data Structure, Distance Emulator.**

I. Introduction

Computing shortest paths is a fundamental problem in network analysis that has been at the core of algorithmic research on graphs for over 60 years. Particular attention has been given to solving this problem on planar graphs, as they naturally appear in many real-world applications, e.g., route planning [24] or image segmentation [21], [34]. Shortest paths are also often utilized as subroutines for solving other important problems on planar graphs such as finding edge separators [1], computing maximum flows [37], [43], [44], [58] and minimum weight cuts [12], [55], or computing distance oracles [17], [20], [32], [56], [57], [61] and planar emulators [15].

We study the *fully dynamic* all-pair shortest path (APSP) problem on planar graphs. Here, the goal is to efficiently maintain a planar graph undergoing edge insertions or deletions (referred to as *edge updates*) and to support distance queries about the shortest path between any source-target vertex pair as quickly as possible. While near-optimal static algorithms for computing shortest paths on planar graphs are known [17], [32], [57], the problem appears to be more challenging in the dynamic setting.

A naive dynamic algorithm simply updates the underlying graph when updates arrive and then runs a static $O(n)$ time SSSP algorithm for answering queries (see [41]). The other extreme would be to process each update by rebuilding a distance oracle on the current graph that then can be queried. This yields update time $n^{1+o(1)}$ and query time $\tilde{O}(1)$ (see [17], [32], [57]). The first sublinear bound on update and query time was obtained in the seminal work by Fakcharoenphol and Rao [27], who showed that non-negative edge weight updates and distance queries can be supported in $O(n^{2/3}\log^{7/3} n)$ time. After a series of follow-up works [44], [45], [49], Gawrychowski and Karczmarz [33] improved the running time to $O(n^{2/3}\frac{\log^{5/3} n}{\log^{4/3}\log n})$, even when allowing negative weights. On the lower bound front, Abboud and Dahlgaard [4] proved there is no algorithm for exact dynamic planar APSP that achieves $O(n^{1/2-\epsilon})$ time for both updates and queries unless the APSP conjecture (on static general graphs) is false.

Moving to approximate distance queries paves the way for circumventing the conditional hardness results, thus obtaining faster algorithms. The first $(1+\epsilon)$-approximation algorithm for fully dynamic planar APSP dates back to the work of Klein and Subramanian [51], who achieved $\tilde{O}(n^{2/3})$ update and query time. Their running time was subsequently improved to $\tilde{O}(\sqrt{n})$ in a breakthrough work by Abraham, Chechik and Gavoille [7] by a reduction to forbidden-set labeling schemes. However, their algorithm cannot break the $\tilde{O}(\sqrt{n})$ barrier on the running time, leaving the following key question open:

Is there a $(1+\epsilon)$-approximate fully dynamic planar APSP algorithm that achieves subpolynomial update time?

Arnold Filtser was supported by the Israel Science Foundation (grant no. 1042/22) and Maximilian Probst Gutenberg was supported by the Swiss National Science Foundation (grant no. 200021 204787).

In this work, we answer this question in the affirmative by developing such an algorithm for the approximate fully dynamic planar APSP problem. Our main result is summarized in the theorem below.

Theorem I.1. *Given an n-vertex undirected planar graph $G = (V, E, w)$ with weights in $[1, W]$ undergoing edge insertions and deletions that preserve the embedding, and a precision parameter $1/\operatorname{polylog}(n) < \epsilon < 1$, there is a* deterministic *data structure that supports edge updates and queries for the $(1+\epsilon)$-approximate distance between any pair s, t of vertices in the current graph G. The data structure can be initialized in $\tilde{O}(n \log W)$ time and achieves $n^{o(1)} \log W$ amortized update and query time.*

Remark I.2. *We can extend the data structure to also support queries for an $(1+\epsilon)$-approximate shortest path from a vertex s and t upon which it returns such a path $\tilde{\pi}_{st}$ in time $n^{o(1)} + O(|\tilde{\pi}_{st}|)$.*

One striking aspect of our result is that it works in the $(1+\epsilon)$-approximate regime, i.e., we can report distances that approximate the shortest path distances within a factor of $(1+\epsilon)$. On general graphs, such results are ruled out since the fully dynamic APSP problem admits strong conditional lower bounds [5], [39] in the low-approximation regime: under plausible complexity assumptions, there is no fully dynamic APSP algorithm with a $(3-\delta)$ approximation ratio that achieves small polynomial update and query times, for any $\delta > 0$. Additionally, for constant ϵ, our running time is faster than the conditional lower bound for exact fully dynamic planar APSP, as discussed above.

A. Related Works

Partially and Offline Dynamic APSP on Planar Graphs.: Dynamic planar APSP has been studied both in incremental and decremental settings. Karczmarz [46] showed a decremental $(1+\epsilon)$-approximate APSP algorithm with $\tilde{O}(n^{3/2}/\epsilon)$ total update time and $\tilde{O}(n)$ query time, even when the underlying planar graph is *directed*. In the incremental setting, Das, Probst Gutenberg, and Wulff-Nilsen [22] designed an *exact* planar APSP algorithm with $\tilde{O}(\sqrt{n})$ worst-case update and query time. In fact, their algorithm extends to the offline setting, where all edge updates and queries are revealed upfront, and achieves the same runtime guarantees, thus matching the conditional lower bound of Abboud and Dahlgaard [4]. Recently, Chang, Krauthgamer, and Tan [15] considered designing $(1+\epsilon)$-approximate algorithms in the offline setting and showed that a running time of $O(\operatorname{poly}\log n)$ can be achieved for any constant ϵ.

Algorithm Engineering.: Due to the importance of understanding the robustness of road networks in route planning applications, fully dynamic APSP has also been extensively studied in the algorithm engineering community [23]–[25], [35], [60]. While these algorithms typically don't offer any theoretical guarantees, they exploit the special structure of road networks, beyond planarity, and are extremely efficient on real-world instances, processing updates in milliseconds for graphs with tens of millions of nodes. In an effort to theoretically explain the impressive performance of such algorithms in practice, Abraham et al. [6] studied a different dynamic model in the $(1+\epsilon)$-approximate setting, allowing arbitrary edge weight updates as long as the shortest path metric of the updated graph is within a factor of M of the shortest path metric of the initial graph. When M is poly-logarithmic and ϵ is a constant, their algorithm achieves poly-logarithmic update and query time.

Fully Dynamic APSP on General Graphs.: Dynamic algorithms for solving APSP on general graphs have received significant attention in the approximate setting [8]–[11], [19], [30], [31], [36], [38], [47], [54], [59], [62]. These results require a very large constant approximation for a small polynomial update time, whereas our result for planar graphs achieves $(1+\epsilon)$-approximation in sub-polynomial time. Moreover, the recent conditional lower bound due to Abboud et al. [3] (and the later refinement by in [2]) shows that on general graphs, there is no fully dynamic algorithm that can simultaneously achieve constant approximation and subpolynomial update and query time. For more trade-offs between the approximation ratio and running times for dynamic APSP on general graphs, we refer the reader to [31], [36], [54] and the references therein.

II. Technical Overview

A. A Dynamic Distance Oracle via Dynamic Emulators for Planar Graphs

The starting point of our story is the recent ε-emulator of Chang, Krauthgamer, and Tan [14]. Given a planar graph $G = (V, E, w)$ and a subset of terminals $T \subseteq V$, we define an *instance* as a tuple (G, T). We say that an instance (H, T) is an ε-emulator of (G, T) if $H = (V_H, E_H, w_H)$ is a planar graph over a set of vertices V_H containing the terminals $T \subseteq V_H$, that preserves distances between terminals up to a small multiplicative factor:

$$\forall u, v \in T : d_G(u, v) \leq d_H(u, v) \leq (1+\varepsilon) \cdot d_G(u, v).$$

The sequence of amazing work [13], [16], [18], [28], [29], [52] developed algorithms to construct an ε-emulator of any given instance (G, T) denoted as (H, T) of size $|V(H)| \leq O\left(|T|, \frac{1}{\varepsilon}\right)$ which is independent of the size of the underlying planar graph G. Following a line of previous work, in a recent breakthrough by Chang et. al. [14] proposed an algorithm to construct ε-emulator (H, T) of any instance (G, T) in $\tilde{O}(\frac{n}{\varepsilon^{O(1)}})$ time with $|V(H)| = \tilde{O}(\frac{|T|}{\varepsilon^{O(1)}})$. The main contribution of this work is to show that the emulator of [14] can be maintained dynamically when the underlying planar graph $G = (V, E)$ does through *edge updates*. This technical result is summarized in the following informal theorem.

Informal Theorem II.1. *Given a dynamic instance (G, T) that is an instance that at each time step either has an edge update to G or a vertex update to T, a size parameter n that upper bounds the number of vertices in G and the guarantee* *that T is of size at most 2 at any time, a target parameter $1 \leq \tau \leq \frac{n}{\log(n)}$, and a precision parameter $0 < \epsilon < 1$.*

Then, there is a deterministic algorithm that maintains explicitly an ε-emulator (H, T) of (G, T) such that

- *Emulator Size: $|H| = \tilde{O}(\sqrt{n\tau}/\epsilon^2)$, and*
- *Emulator Recourse: each update to (G, T) can be processed such that the amortized number of changes to H is $\tilde{O}(1/\epsilon^2)$, and*
- *Update Time: each update to (G, T) can be processed in amortized time $\tilde{O}\left(\frac{n}{\epsilon^2 \tau}\right)$.*

The algorithm takes initialization time $\tilde{O}(n/\epsilon^2)$.

In our *dynamic distance oracle data structure*, we aim to use the above theorem to maintain a small emulator of (G, T) where we let T be the empty set. Then, whenever the adversary issues a query for vertices $s, t \in V$, we simply update the set T by adding s, t to the formerly empty set. We then compute the distance from s to t in the updated small emulator (using a single-source shortest paths algorithm), return the distance as an estimate for the st-distance in G, and finally delete s, t again from set T.

However, note that Informal Theorem II.1 only achieves subpolynomial update bounds for target parameter τ being subpolynomially close to n. Fortunately, since the emulator has extremely low recourse (only polylogarithmic), we can resolve this problem by building a hierarchy. We simply invoke Informal Theorem II.1 again on the emulator to obtain a slightly smaller emulator, and so on until we obtain an emulator of subpolynomial size as desired. Due to the small recourse bounds, we are able to control the amortized update time spent on updating each emulator by only a slightly larger sub polynomial factor. This yields our dynamic distance oracle algorithm given in Theorem I.1.

The remainder of this overview sketches the algorithm and proof ideas to obtain Informal Theorem II.1.

B. A Brief Review of [14]

Since our algorithm crucially builds on the work of Chang et. al. [14], let us briefly discuss their strategy to computing an ε-emulator (H, T) for an instance (G, T). For the rest of the section, we fix $1 \leq \tau \leq \frac{n}{\log^D(n)}$ for large enough constant $D > 1$.

The emulator construction by Chang et. al. [14] follows a divide-and-conquer framework. To keep track of the divide-and-conquer step, they use a decomposition tree $\mathcal{T}$. Initially, the tree $\mathcal{T}$ only contains a single root node that is associated with the input instance (G, T). Henceforth, since each node in $\mathcal{T}$ is associated with an instance, we use nodes and their associated instances interchangeably.

The algorithm then recursively initializes an instance (R, S) in $\mathcal{T}$ as follows:

- If the instance (R, S) is of the size $|V(R)| \leq O(n/\tau)$, S of the size $\mathrm{polylog}(n)$, and all terminals in S on a single face of R, then it labels (R, S) a leaf node.
- Otherwise, it breaks the graph R into subgraphs $R_1, R_2, \ldots, R_k$ finds a *portal set* $\textsc{Portals}_{(R,S)}$ (whose role will be clarified in a moment), and adds for every $0 \leq i \leq k$, the instance (R_i, S_i) where $S_i = (S \cup \textsc{Portals}_{(R,S)}) \cap V(R_i))$ as a child of (R, S) to $\mathcal{T}$. Note here in particular, that the portals of an instance become terminals to the child instances. In addition, the algorithm ensures that the vertices $\textsc{Portals}_{(R,S)}) \cap V(R_i))$ lies on the *boundary* of the planar graph R_i w.r.t. R; i.e. vertices with the incident edges in R that lies in the edge set of R_j for $j \neq i$.

When constructing the emulator (H, T) for (G, T), [14] follows a bottom-up approach: for every leaf (R, S) in $\mathcal{T}$, they can construct a 0-emulator (H_R, S) (that is preserving distances in R exactly) with H_R of size at most $O(|S|^4) \leq \mathrm{polylog}(n)$ [52].

For any internal node (R, S) in $\mathcal{T}$, they obtain the emulator (H_R, S) by "glueing" the emulators of its children, i.e. the emulators $(H_{R_1}, S_1), (H_{R_2}, S_2), \ldots, (H_{R_k}, S_k)$, in the vertices in S and $\textsc{Portals}_{(R,S)}$.

The game, and the source of tension in this algorithm is the following: one has to break the instance (R, S) into sub-instances $(R_1, S_1), (R_2, S_2), \ldots, (R_k, S_k)$, including choosing portals $\textsc{Portals}_{(R,S)}$ along the boundaries of these sub-instances such that simultaneously it satisfies the following properties:

- The number of new portals should not blow up since the size of the emulator is at least as large as the total number of terminal vertices in the leaves of $\mathcal{T}$,
- The resulting emulator of (H_R, S) from gluing the emulators of $(H_{R_1}, S_1), (H_{R_2}, S_2), \ldots, (H_{R_k}, S_k)$ along the portals have small *stretch* in the distances between terminals since the overall stretch can accumulate as built emulator of (G, T) in bottom-up fashion, and
- Each child instance (R_i, S_i) has either
  - R_i of significantly smaller size than R,
  - S_i of significantly smaller size than S, or
  - S_i being incident to fewer faces in R_i than S in R.

The last property then ensures that the depth of the decomposition tree $\mathcal{T}$ is bounded logarithmically in the size of the graph G.

This game is successfully resolved in [14] by decomposing carefully by case distinction. We delve into more detail on their construction in the next section.

C. Initialization of our Emulator Data Structure

In this section, we describe the initialization procedure of our algorithm to obtain the initial decomposition tree $\mathcal{T}$. Our construction closely follows the emulator construction in [14].

Before we delve into the algorithm, the astute reader might note that initially, the input instance (G, T) has $T = \emptyset$. Thus, the resulting emulator does not need to preserve any distances (as there are no terminals). However, here we compute the tree decomposition and the emulator since we can show that when we update T by two vertices, we only need a few updates to the emulator to preserve the distances between vertices in T.

We point out that in our construction of $\mathcal{T}$, we have that for each instance $(R, S) \in \mathcal{T}$, we have that all terminals S are on the *holes* of R with respect to G. Here we define a hole to be a face of R that is not a face of G. We henceforth call an instance (R, S) an h-hole instance if R has at most h holes.

Recall that the goal when constructing decomposition tree $\mathcal{T}$ is to break the graph into one-hole instances of size at most $O(\frac{n}{\tau})$ with at most $\mathrm{polylog}(n)$ terminals. We call such an instance **Type-1** instance. Type-1 instances will be the leaves of our hierarchical decomposition tree $\mathcal{T}$. For Type-1 instances we will simply construct a 0-emulator of size $\mathrm{polylog}(n)$ in $O(\frac{n}{\tau})$ time [52].

In order to get to Type-1 instances we will gradually break the graph, where there will be 4 types of instances. The instances will respect the hierarchy of the tree. That is a parent instance will always be of equal or higher Type-than a child instance. We now elaborate on the different types:

- **Type-4:** An instance (R, S) is Type-4 if it has more than $\frac{n}{\tau}$ vertices and at most a constant number of holes. Note that the initial instance $(G, \emptyset)$ is of Type-4. A vertex $v \in R$ is a *boundary vertex* w.r.t. the instance (R, S) if it is incident to an edge $e \notin R$. Type-4 instances have the special property that the terminal set S will simply be the boundary vertices. To break a Type-4 instance we will use the planar separator of Klein et. al. [50] (see Theorem III.7). Specifically, [50] receives an ℓ-hole instance (R, S), where S is the set of boundary vertices. The algorithm then partitions the instance (R, S) into sub-instances $(R_0, S_0), \dots, (R_q, S_q)$, where each edge belongs to a single subgraph (a vertex might belong to several), each instance has size at most $|R_i| \le \frac{3}{4} \cdot |R|$, the total number of boundary vertices is bounded by $\sum_{i=0}^{q} |S_i| = O(\sqrt{|R|})$, and each instance has at most $\frac{3}{4}\ell + 4$ holes (in our case, an absolute constant). Finally, we let $\textsc{Portals}_{(R,S)} = \cup_i S_i \setminus S$.
- **Type-3:** An instance (R, S) is Type-3 if it has at most $O(\frac{n}{\tau})$ vertices and a constant number of holes.[1] Here our goal is to reduce the number of holes. The way we reduce the number of holes is fairly standard. We find a shortest path π_{uv} between vertices u, v laying on different holes h, h'. We then duplicate the path π_{uv} (including all vertices and edges), and "slice" a new face including the holes h, h' and the two copies of π_{uv} (see Figure 1). As a result, the number of holes is reduced by 1. Denote the resulting graph by R'. We then turn some of the vertices along π_{uv} into portals. Specifically, for every terminal $x \in S$, using [48], [61] (see Lemma III.12) one can pick a set P_x of $O(\frac{1}{\varepsilon})$ vertices along π_{uv} such that for every vertex $z \in \pi_{uv}$ on the path, there is a nearby vertex $y \in P_x$: $d_{R[\pi_{uv}]}(y, z) \le \varepsilon \cdot d_R(x, z)$. The vertices in P_x can be used to re-route every path starting at x that crosses π_{uv} while introducing only a small $1 + O(\varepsilon)$-stretch. We thus set $\textsc{Portals}_{(R,S)} = \bigcup_{x \in S} P_x$. The instance (R, S) will have a single child $\left(R', S \cup \bigcup_{x \in S} P_x\right)$. The fact that enables this reduction to work is that given a planar δ-emulator $\tilde{R}'$ for $\left(R', S \cup \textsc{Portals}_{(R,S)}\right)$, we can identify the two copies of each portal in $\textsc{Portals}_{(R,S)}$ along π_{uv} to obtain an $O(\varepsilon + \delta)$-emulator $\tilde{R}$ for (R, S).
- **Type-2:** An instance (R, S) is Type-2 if it is one-hole and has at most $O(\frac{n}{\tau})$ vertices. The purpose of Type-4 and Type-3 was to reduce to Type-2 instances. Here we break the instance into two Type-2 instances $(R_1, S_1), (R_2, S_2)$ such that each of S_1, S_2 is smaller than S by a constant factor. We proceed by finding a shortest path π_{uv} where u, v are pair of vertices laying on the hole of R, such that π_{uv} partitions R into two subgraphs R_1, R_2, each containing at most $\frac{11}{12} \cdot |S|$ terminals (see Theorem III.10). We will duplicate the path π_{uv} for each of the two child instances. Surprisingly, as was shown by [14],[2] it is possible to pick a set P_π of $O(\frac{1}{\varepsilon} \cdot \log n)$ portals along π_{uv} such that for every vertex x on the hole (not necessarily a terminal), and $z \in \pi_{uv}$ on the separator path, there is a nearby portal $y \in P_\pi$: $d_{R[\pi_{uv}]}(y, z) \le \varepsilon \cdot d_R(x, z)$. The vertices in P_π can be used in order to re-route every path starting on the hole and crossing π_{uv} while introducing only a small $(1 + O(\varepsilon))$-stretch [15][Small Spread Case].Thus again, we take $\textsc{Portals}_{(R,S)} = P_\pi$. We then recursively execute the two one-hole instances $\left(R_1, (S \cap R_1) \cup \textsc{Portals}_{(R,S)}\right)$, $\left(R_2, (S \cap R_2) \cup \textsc{Portals}_{(R,S)}\right)$.

a) Depth of $\mathcal{T}$.: Let us now more formally bound the depth of the hierarchical decomposition tree $\mathcal{T}$. Consider a root $(G, \emptyset)$ to leaf path $\mathcal{P}$ in $\mathcal{T}$. For each Type-4 instance (R, S) it is guaranteed that each child instance (R_i, S_i) has size at most $|R_i| \le \frac{3}{4} \cdot |R|$. Therefore there will be at most $O(\log n)$ Type-4 instances along $\mathcal{P}$. A Type-3 instance has a constant number of holes, and the number of holes is reduced by 1 in each iteration. Therefore there will be only a constant number of Type-3 instances along $\mathcal{P}$. For a Type-2 instance (R, S) it is guaranteed that each child instance (R_i, S_i) has at most $|S_i| \le \frac{11}{12} \cdot |S|$ terminals. Therefore there will be at most $O(\log n)$ Type-2 instances along $\mathcal{P}$. Type-1 instances have no children. Overall, the depth of $\mathcal{T}$ is bounded by $O(\log n)$.

b) Number of Copies.: While there is no bound on the number of instances $(R, S) \in \mathcal{T}$ a vertex might belong to, an edge e can belong to at most $O(\log^2 n)$ instances. Indeed, in our treatment of Type-4 instances, no edge is ever duplicated. In a Type-3 instance, we duplicate a single shortest path (which contains no edges lying on a hole). Note that if an edge e is duplicated, then both its copies will now be laying on a hole and will be duplicated no more (in Type-3 instances). Type-2 instances are the most tricky. Here we pick a separator path π_{uv} and duplicate all the edges along it. Note that once an edge e is duplicated in such a way, the edge will be laying on the single hole of each instance it belongs to. Consider a Type-2 instance (R, S) and an edge e on its hole. Let π_{uv} be the separator path, and $(R_1, S_1), (R_2, S_2)$ be the two child

[1]It is possible that a child of a Type-3 instance will have more vertices that its parent. We therefore allow slack in the upper bound on the number of vertices for Type-3 instances.

[2]For this statement to hold we will assume that all the edge weights are in $[1, n^4]$.

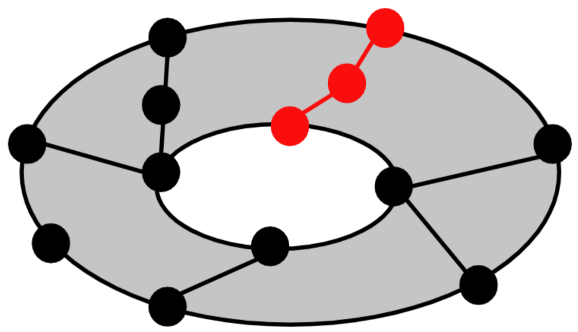
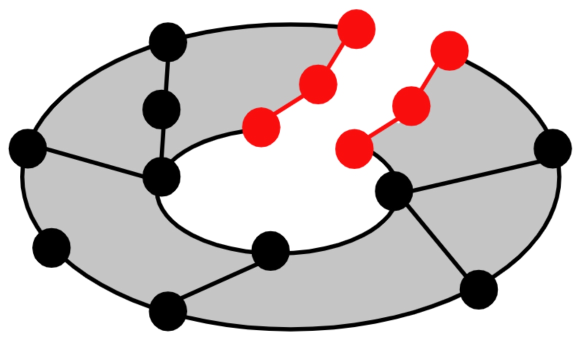


Fig. 1: Illustration of our procedure for a Type-3 instance. On the left, we have the initial instance (R, S) which is incident to two holes (the faces with white background). The algorithm then chooses a shortest path π_{uv} between these two holes (drawn in red). The child instance is obtained by slicing along the entire path. This results in a one-hole instance (R_1, S_1) (on the right) as the slicing connects the two holes.

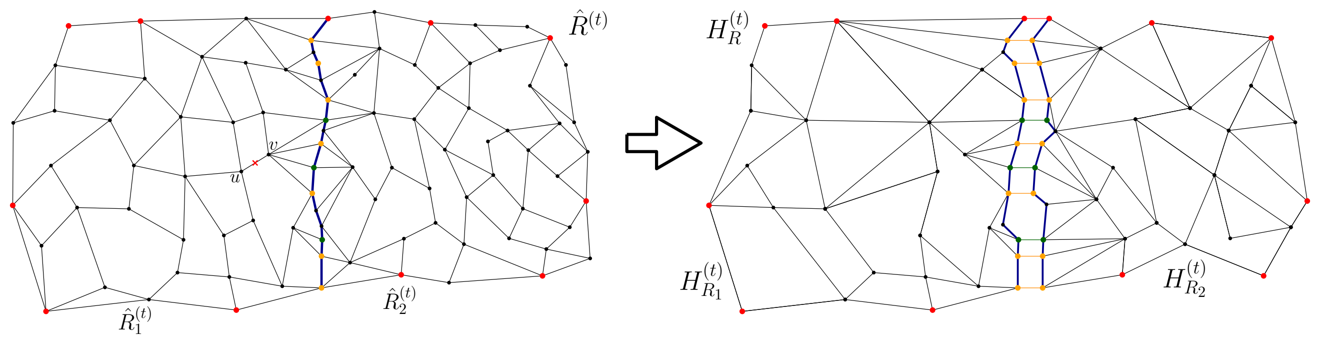


Fig. 2: On the left $(R^{(t)}, S^{(t)})$ is a Type-2 instance. The red vertices represent terminals. The blue path π is a balanced separator in R. The orange vertices along π are the original portals P_π. At some point the edge $\{u, v\}$ is deleted. As a result, the green portals are added to π. On the right are emulators $H^t_{R_1}$ and $H^t_{R_2}$ that been constructed for the instances $(R_1^{(t)}, \hat{S}_1^{(t)})$ and $(R_2^{(t)}, \hat{S}_2^{(t)})$ respectively. We construct an emulator H^t_R for $(R^{(t)}, \hat{S}^{(t)})$ by "gluing" the two emulators using the portals along π.

instances of (R, S). π_{uv} is allowed to include edges laying on the hole, and in particular e. The key observation is that if such an edge e belongs to π_{uv}, then necessarily e will be a *bridge* edge in either (R_1, S_1) or (R_2, S_2). A bridge edge is an edge whose removal disconnects the graph. Theorem III.10 guarantees that the separator path π_{uv} is either a single bridge edge, or does not contain any bridge edges. If π_{uv} is a bridge edge e', we simply delete e' and the child instances are the connected components: $(R_1, S \cap (R_1 \cup e'))$, $(R_2, S \cap (R_2 \cup e'))$. In particular, e' will belong to no strict descendent of (R, S) in $\mathcal{T}$. It follows that a bridge edge is never duplicated. Accordingly, if an edge e belonging to the hole is duplicated and belongs to both $(R_1, S_1), (R_2, S_2)$, then in one of them e has to be a bridge edge and will be duplicated no more (See Figure 3). For the other children, the edge stays on its hole. So inductively applying the same argument, the edge can be duplicated at most $O(\log n)$ times. In addition, all the Type-2 nodes that contain a particular duplicate of the edge lie on a single root-to-leaf path in $\mathcal{T}$. It follows that overall each edge can belong to at most $O(\log^2 n)$ different instances (counting all its different copies).

D. Maintaining the Decomposition Tree

In our dynamic algorithm, we let $\mathcal{T}$ consist of *static* instances, that is, we add instances (R, S) to $\mathcal{T}$ and then do not update them in any form until we delete them again from $\mathcal{T}$. When we delete a node, we delete the entire subgraph rooted at the node from $\mathcal{T}$.

Instead, we define for each such instance $(R, S) \in \mathcal{T}$ the graph $\hat{R}$ which captures how R evolves when applying the updates to (G, T) to (R, S). Let us now formally define $\hat{R}$.

We define $\hat{R}$ recursively. Letting t_R denote the time that (R, S) was added to $\mathcal{T}$, we have that $\hat{R}^{(t_R)} = R$ (we use superscripts to indicate the value of a variable at the time specified by the superscript). For any time $t > t_R$, we define $\hat{R}^{(t)}$ by considering the t-th edge update to (G, T) (if there is one). We say the edge update $e = \{u, v\}$ is *valid* (see Figure 4) if both $u, v \in \hat{R}^{(t-1)}$ and either:

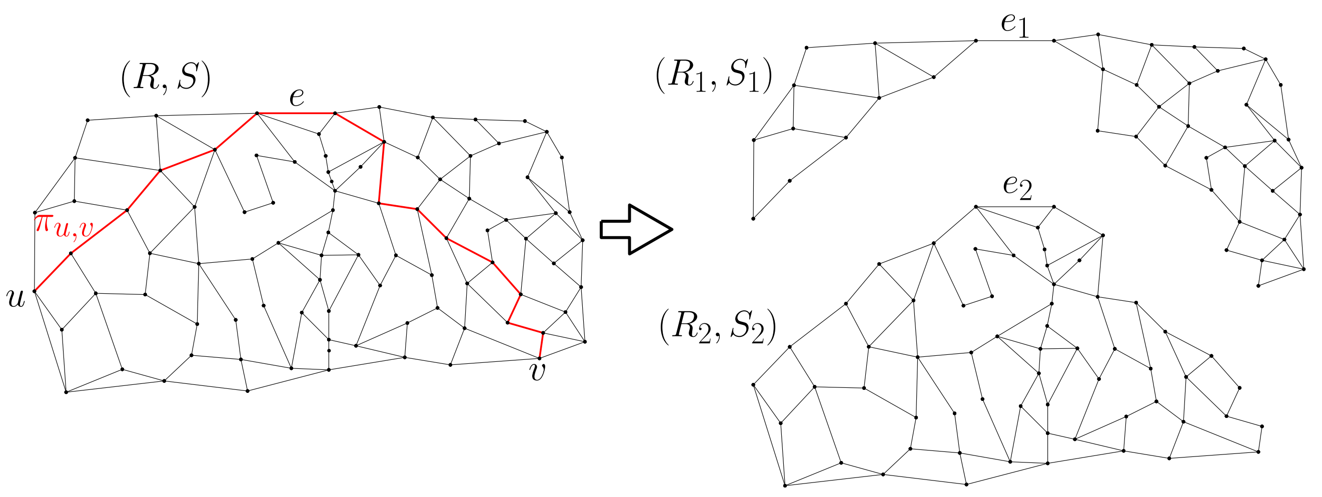


Fig. 3: On the left illustrated a Type-2 instance (R, S) with a balanced separator path π_{uv} colored in red. The edge $e \in \pi_{uv}$ lays on the hole of R. e its duplicated to both R_1, R_2, and became a bridge edge in R_1.

- This is an edge deletion and e is in $\hat{R}^{(t-1)}$.
- This is an edge insertion, and the edge e lies inside a face of $\hat{R}^{(t-1)}$ which is not a hole. We stress that the validity of an edge update is defined w.r.t. $\hat{R}^{(t-1)}$ and not R. Note that the set of faces of $\hat{R}^{(t-1)}$ can undergo considerable changes compared to R. Consider for example a Type-2 instance (R, S). A deletion of an edge e laying on the hole of R will incorporate another face f to be part of the hole. Consider now an insertion update of an edge e' which is internal to the face f. If the insertion update of e' happened before the deletion of e, then e' is a valid edge update w.r.t. (R, S). Otherwise, if first e was deleted, and only then e' inserted, then e' is part of the hole and is not a valid edge update any longer (see Figure 4).

If an edge update is valid, we apply it to $\hat{R}^{(t-1)}$ to obtain $\hat{R}^{(t)}$. Otherwise, we simply let $\hat{R}^{(t)}$ be equal to $\hat{R}^{(t-1)}$.

Note that every edge update is a valid edge update to the root instance $(G, \emptyset)$ of $\mathcal{T}$ (as $G^{(t)}$ has no holes with respect to itself). Note also that being a valid edge update is "hereditary": if e is a valid edge update in $\hat{R}^{(t)}$, then e is also a valid edge update in all the ancestors of $\hat{R}^{(t)}$ in $\mathcal{T}$. We say that an instance (R, S) is a minimal valid instance w.r.t an edge update e, if e is a valid edge update w.r.t. (R, S), but not a valid edge update w.r.t. any of its children (in particular if (R, S) is Type-1). Note that a deletion of an edge e is a valid w.r.t. all instances including e. On the other hand, consider an edge update of inserting the edge e into a face f. As faces are never duplicated in our framework, such an edge update can have at most a single minimal instance.

Finally, for each instance (R, S), we also define $\hat{S}$ to be the set obtained as the union of S and the portal sets of all strict ancestor instances of (R, S) in $\mathcal{T}$.

Given this definition of $\hat{R}$ and $\hat{S}$ for every instance (R, S), we can first give our update procedure to update the sets $\text{PORTALS}_{(R,S)}$. This procedure is given in Algorithm 3 and adds portals upon an edge update e to (G, T) as follows:

- For every minimal valid instance (R, S), we add the endpoints of e as new portals to $\text{PORTALS}_{(R,S)}$.
- For every valid instance (R, S) of Type-2 or 3 (not necessarily minimal), we will add some new portals to ensure the stretch guarantee of the resulting emulators (as distances might change). Consider the relevant path π_{uv} for the instance (the separator path for Type-2 instances, and the shortest path between two holes for Type-3 instances). For each vertex $z \in e = \{a, b\}$, using [48], [61] (see Lemma III.12) we will pick a set P_z of $O(\frac{1}{\varepsilon})$ new portals along π_{uv} such that for every vertex $x \in \pi$ on the path, there is a nearby portal $y \in P_z$: $d_{\hat{R}[\pi]}(y, x) \leq \varepsilon \cdot d_{\hat{R}}(x, z)$. Importantly, the set P_z is computed w.r.t. the original instance $R = R^{t_R}$ at the initialization time (and not w.r.t. the current instance $\hat{R}^{(t-1)}$). This is crucial as [48], [61] assumes π to be a shortest path. The sets $P_u \cup P_v$ will be added as new portals to $\text{PORTALS}_{(R,S)}$. The surprising part is that even though we pick the portals w.r.t. a path π which might be severely damaged, and no longer be a shortest path, this is still good enough for the stretch guarantee. The reason, as we will see later, is that if there has been another change preventing us from using the new dedicated portals, then this change, in turn, introduced new portals to take care of this issue.

Finally, we monitor for every instance $(R, S) \in \mathcal{T}$ whether R is close to $\hat{R}$ and S is close to $\hat{S}$. If the difference in either set is too large, we delete (R, S) from $\mathcal{T}$ (along with the subtree rooted at it) and then add in the same place, the instance $(\hat{R}, \hat{S})$ which we then initialize using our static procedure.

Before we move to the next section, we mention a subtle but crucial point: we already sketched a proof that every edge is only contained in $O(\log^2 n)$ instances in $\mathcal{T}$ at any point in time, which means that the above update procedure can be implemented in $\tilde{O}(1)$ time. But, equally crucial, we can also

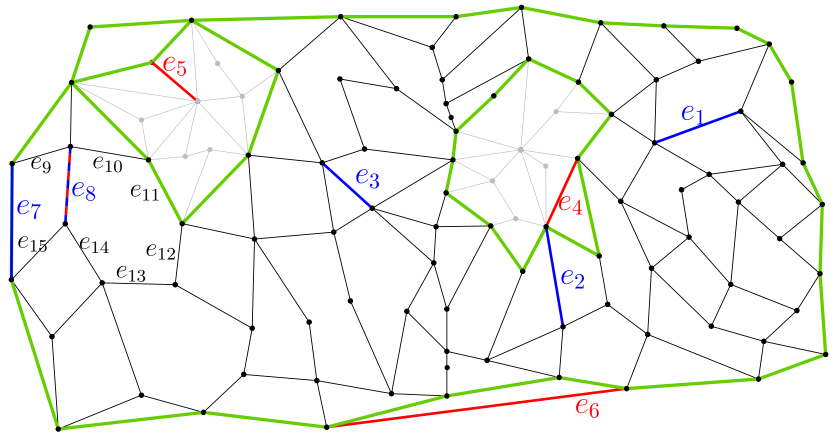


Fig. 4: There are three holes encircled by green edges. The edges e_1, e_2, e_3 are valid edge updates (insertion or deletion). The edges e_4, e_5, e_6 are invalid edges updates as they lay inside a hole. e_7 is a valid edge update (deletion). The edge update of inserting e_8 here is somewhat tricky. If we first insert e_8 and then delete e_7, both updates are valid. From the other hand, if we first delete e_7, then the edges e_9 to e_{15} become part of the hole. In particular, inserting e_8 is no longer a valid edge update.

update within the same asymptotic time bounds all graphs $\hat{R}$ and sets $\hat{S}$. This uses that every vertex only occurs as a non-terminal vertex in at most $O(\log^2 n)$ instances, and thus only has to be added to at most $O(\log^2 n)$ sets $\hat{S}$ when added to a portal set (recall that $\hat{S}$ is grown only by adding vertices to one of its ancestors portal sets).

E. Emulator Construction and Maintenance: Size, Update time, and Recourse

In this section, we discuss the construction and maintenance procedure of the emulator (H, T) for (G, T). As previously mentioned, the emulator is constructed in a bottom-up manner. Here, we focus in particular on bounding the recourse caused by each update to (G, T) as this is key to obtaining an efficient emulator hierarchy which then allows us to glean off distances efficiently.

a) Type-1 Emulators.: For a Type-1 instance (R, S), we simply compute a 0-emulator $(H_{\hat{R}}, \hat{S})$ on the graph $\hat{R}$ with terminals $\hat{S}$. Using the algorithm from [52], we can obtain such a 0-emulator with size $O(|\hat{S}|^4) = O(|S|^4) = \text{polylog}(n)$ in $O(|V(R)| \cdot |\hat{S}|) = O(|V(R)| \cdot |S|) = \tilde{O}(|V(R)|) = \tilde{O}(\frac{n}{\tau})$ time. Whenever $\hat{R}$ or $\hat{S}$ change, the emulator $(H_R, \hat{S})$ is computed from scratch. The update (and computation) time is $\tilde{O}(\frac{n}{\tau})$, while the recourse is $\text{polylog}(n)$.

b) Type-2 Emulators.: Next, consider a Type-2 instance (R, S) with children $(R_1, S_1), (R_2, S_2)$ and a separator path π_{uv} duplicated in both child instances.[3]

We then build the emulator as the union of $(H_{\hat{R}_1}, \hat{S}_1)$ and $(H_{\hat{R}_2}, \hat{S}_2)$ glued in the vertices $\hat{S}$ and additionally add all edges added to $\hat{R}$ since initialization time for which (R, S) was the minimal valid instance. This yields $(H_{\hat{R}}, \hat{S})$. Figure 2 illustrates the high-level idea of the gluing procedure.

We show that the size of $H_{\hat{R}}$ at initialization time is $|S| \cdot \text{polylog}(n)$.[4] By forcing a rebuild of (R, S) (i.e. removing (R, S) from $\mathcal{T}$ and replacing it by $(\hat{R}, \hat{S})$) once $|\hat{S}| = (1 + \Theta(1/\log^2(n)))|S|$, we can ensure that the size of $H_{\hat{R}}$ does not increase beyond $O(|S| \cdot \text{polylog}\,|S|)$. The recourse is at most polylogarithmic per update as can be established by induction and the fact that $\hat{S}$ increases by at most $O(\log^2 n/\epsilon)$ vertices per time step and since we can amortize the recourse when rebuilding (R, S) over the many changes to $\hat{S}$ that it took in the meantime.

Finally, we also point out that we rebuild (R, S) whenever $\hat{R}$ underwent $\tilde{\Theta}(|S|)$ changes. Again, recourse can be amortized for by updates, and this is in fact necessary as it upper bounds the number of edges for which (R, S) is a minimal instance by $\tilde{O}(|S|)$ which in turn is necessary to bound the size of the emulator.

c) Type-3 Emulators.: Next, consider a Type-3 instance (R, S) with a single child[3] (R', S') and a path π_{uv} in R which is duplicated in R'. In a similar fashion to Type-2 instances, the emulator $(H_{\hat{R}}, \hat{S})$ for $(\hat{R}, \hat{S})$ is obtained from the emulator $(H_{\hat{R}'}, \hat{S}')$. Again, we rebuild only after many changes to $\hat{S}$. This yields an analogous analysis as for Type-2 emulators.

d) Type-4 Emulators.: Finally, we update the emulator for Type-4 instances trivially: given a Type-4 instance (R, S) with children $(R_1, S_1), (R_2, S_2), \ldots, (R_k, S_k)$, we simply take $(H_{\hat{R}}, \hat{S})$ to be the union of emulators

[3]At construction time, Type-2 instances will always have two children, while Type-3 instances will have a single child. However, at re-computation time, it might be that the instance became disconnected (due to edge deletions), and as a result, might have more children. We will ignore these cases in this overview.

[4]The subtree of $\mathcal{T}$ rooted at (R, S) is a full binary tree of depth $\ell = O(\log |S|)$, where the leaves are Type-1 instances, each with at most $k = \log^{10} n$ terminals. We can give each terminal $(1 + O(\frac{\log n}{k \cdot \varepsilon}))^{\ell} \cdot k^3 = \text{polylog}(n)$ coins. At height i in $\mathcal{T}$, each terminal should have $(1+O(\frac{\log n}{k \cdot \varepsilon}))^{i} \cdot k^3$ coins, then we create $O(\frac{\log n}{\varepsilon})$ new portals along the path π and there are enough coins to give the new terminals and the remaining ones. Eventually in the leaves each terminal will have k^3 coins which are enough to pay for the $|S|^4$ size emulator.

$(H_{\hat{R}_1},\hat{S}_1),(H_{\hat{R}_2},\hat{S}_2),\dots,(H_{\hat{R}_k},\hat{S}_k)$ and the set of edges that were inserted and for which R was the minimal valid instance.

Again, we rebuild (R,S) when $\hat{R}$ underwent many changes. The analysis is similar to Type-2 instances and therefore omitted.

F. *Stretch analysis*

Consider an instance $(R,S)\in\mathcal{T}$ at time $t\ge t_R$. $\hat{R}^{(t)}$ denotes the graph at time t, which is R when we taking into account also all the valid edge updates. $\hat{S}^{(t)}$ denotes the set of terminals in $\hat{R}^{(t)}$, these are portals inherited from the parent (original and new). Denote by $U_R^{(t)}$ the subset of $\hat{R}^{(t)}$ vertices that been incident to any edge update since t_R, or to vertices that became new terminals (that is $\hat{S}^{(t)}\setminus S\subseteq U_R^{(t)}$). Initially $U_R^{(t_R)}=\emptyset$. Let $H_R^{(t)}$ denote the emulator of $\hat{R}^{(t)}$. Suppose that (R,S) is at height ℓ in $\mathcal{T}$. We will argue that $H_R^{(t)}$ has stretch $(1+O(\varepsilon))^\ell$ w.r.t. $\hat{S}^{(t)}\cup U^{(t)}$. That is: $\forall u,v\in\hat{S}^{(t)}\cup U^{(t)},\ d_{\hat{R}^{(t)}}(u,v)\le d_{H_R^{(t)}}(u,v)\le(1+O(\varepsilon))^\ell\cdot d_{\hat{R}^{(t)}}(u,v)$. In particular, the emulator $H_G^{(t)}$ of G will have stretch $(1+\varepsilon)^{O(\log n)}$. Later we will reduce the stretch by scaling ε appropriately.

The proof is by induction on the height in $\mathcal{T}$. The base case is Type-1 instances, where the stretch is indeed 1, as we use 0-emulators (and all the vertices in $\hat{S}^{(t)}\cup U^{(t)}$ are portals). The proof for Type-4 instances is quite straightforward and we will discuss it briefly at the end. Type-2-3 instances are more challenging and lay in the heart of our analysis. We will focus on Type-2 instances in our discussion here. Consider a Type-2 instance $(\hat{R}^{(t)},\hat{S}^{(t)})$ at height ℓ in $\mathcal{T}$. At initialization time t_R, we chose a separator path π, and used it to create child instances $(R_1,S_1),(R_2,S_2)$, where π been part of their hole. In addition, we chose a set P_π of portals along π that became terminals in R_1,R_2. As the algorithm progressed, R added more portals along π. We denote by $P_\pi^{(t)}$ the set of portals on π added by R up to time t. Note that $P_\pi^{(t)}\cap(V(\hat{R}_1^{(t)})\cap V(\hat{R}_2^{(t)}))\subseteq\hat{S}_1^{(t)}\cap\hat{S}_1^{(t)}$. The following invariant is always kept:

Invariant. *For every vertex $x\in S\cup U_R^{(t)}$ and $y\in\pi$ on the separator, there is a portal $y'\in P_\pi^{(t)}$ such that $d_{\hat{R}^{(t)}[\pi]}(y,y')\le\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$.*

In words, the invariant ensures us that there is always a portal $y'\in P_\pi^{(t)}$ such that the induced shortest path from y to y', that is the shortest path using only original π edges, is of length at most $\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$. Note that the invariant holds for all the vertices in $\hat{S}^{(t)}$, as $\hat{S}^{(t)}\setminus S\subseteq U_R^{(t)}$. At initialization time t_R, P_π is chosen exactly so that the invariant will hold (w.r.t. S, as $U_R^{(t_R)}=\emptyset$). The fact that such a set P_π of $O(\frac{\log n}{\varepsilon})$ portals even exist is one of the key contributions of [14]. We argue that the invariant is maintained after each edge update by induction on the time steps. Suppose that the invariant holds in $\hat{R}^{(t-1)}$, and consider the edge update $e=\{u,v\}$ received at time t. Consider $x\in S\cup U_R^{(t)}$ and $y\in\pi$.

We first assume that x is not an endpoint of e, and that e is valid w.r.t. $\hat{R}^{(t-1)}$ (in particular $x\in(S\cup U_R^{(t-1)})$. Denote by $y'\in P_\pi^{(t-1)}$ the portal fulfilling the invariant in $\hat{R}^{(t-1)}$, that is $d_{\hat{R}^{(t-1)}[\pi]}(y,y')\le\varepsilon\cdot d_{\hat{R}^{(t-1)}}(x,y)$. There are several cases to consider:

- Suppose that e is an edge deletion, and that e does not lay on π. The path from y to y' along π is still a part of $\hat{R}^{(t)}$, while the distance from x to y could only grow as a result of the deletion. Thus the invariant still holds.
- Suppose that e is an edge deletion, and that e lays on π. If e does not lay on the π-path from y to y', then clearly, as in the first case, the invariant still holds. Otherwise, e lays on the π-path from y to y'. Let $u\in e$ be the closer vertex to y on this path. Then clearly, as u joins $P_\pi^{(t)}$, the invariant still holds: $d_{\hat{R}^{(t)}[\pi]}(y,u)\le d_{\hat{R}^{(t-1)}[\pi]}(y,y')\le\varepsilon\cdot d_{\hat{R}^{(t-1)}}(x,y)\le\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$ (the distance from x to y might only increased).
- Suppose that e is an edge insertion, and the shortest $x-y$ path in $\hat{R}^{(t)}$ remain unchanged. Clearly, the distance still holds.
- Suppose that $e=\{u,v\}$ is an edge insertion, and the shortest $x-y$ path in $\hat{R}^{(t)}$ changed. Denote by $Q_{x,y}$ the new $x-y$ shortest path in $\hat{R}^{(t)}$. Necessarily $Q_{x,y}$ goes though e. Suppose w.l.o.g. that $Q_{x,y}$ goes from x to u, then to v and from there to y. As a result of this edge update, we added to $P_\pi^{(t)}$ a new set of portals Q_v. This set of portals is added w.r.t. the original instance (R,S) and insures that for every vertex $z\in\pi$, there is a portal $z'\in P_v$ such that $d_{R[\pi]}(z,z')\le\varepsilon\cdot d_R(v,z)$. In particular, for our $y\in\pi$ there is $y_v\in P_v$ such that $d_{R[\pi]}(y,y_v)\le\varepsilon\cdot d_R(v,y)$. If the path along π from y to y_v remain unchanged, and the length of the shortest path from v to y remain unchanged (or only increased) in $\hat{R}^{(t)}$ (compared with R), then it holds that $d_{\hat{R}^{(t)}[\pi]}(y,y_v)\le\varepsilon\cdot d_R(v,y)\le\varepsilon\cdot d_{\hat{R}^{(t)}}(v,y)\le\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$. Otherwise, there are two cases to consider:
  - $d_{\hat{R}^{(t)}}(v,y)<d_R(v,y)$: The length of the shortest path from v to y decreased in $\hat{R}^{(t)}$ (compared with R). Then this path must go through a vertex v' which been incident to some edge update. In particular $v'\in U_R^{(t-1)}$. By induction there is a portal $y_{v'}\in P_\pi^{(t-1)}\subseteq P_\pi^{(t)}$ such that $d_{\hat{R}^{(t)}[\pi]}(y,y_{v'})=d_{\hat{R}^{(t-1)}[\pi]}(y,y_{v'})\le\varepsilon\cdot d_{\hat{R}^{(t-1)}}(v',y)=\varepsilon\cdot d_{\hat{R}^{(t)}}(v',y)\le\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$.
  - $d_{\hat{R}^{(t)}}(v,y)\ge d_R(v,y)$ and some edges along the sub-path of π from y to y_v have been deleted. Let u be the closest endpoint to y of a deleted edge on the sub-path of π from y to y_v. As necessarily $u\in P_\pi^{(t)}$, it follows that $d_{\hat{R}^{(t)}[\pi]}(y,u)\le d_{R[\pi]}(y,y_v)\le\varepsilon\cdot d_R(v,y)=\varepsilon\cdot d_{\hat{R}^{(t)}}(v,y)\le\varepsilon\cdot d_{\hat{R}^{(t)}}(x,y)$.

Finally, there are the cases where $x\in U_R^{(t)}$ is incident to the new updated edge, or when e is an invalid edge update w.r.t. $\hat{R}^{(t-1)}$ (but still might introduce new terminals). These cases are treated in a similar fashion to the cases we discussed above, and we will skip them in this overview.

Next we consider the emulator $H_R^{(t)}$. $H_R^{(t)}$ contains all the edges (and vertices) of both $H_{R_1}^{(t)},H_{R_2}^{(t)}$. Further, the vertices

$\hat{S}^{(t)}, P_\pi^{(t)}, \hat{S}_1^{(t)}, \hat{S}_2^{(t)}, U_R^{(t)}$ are all in $H_R^{(t)}$ (of course, some of them are identified with each other). This includes vertices in $U_R^{(t)}$ which are not portals in either of $\hat{R}^{(t)}, \hat{R}_1^{(t)}, \hat{R}_2^{(t)}$. This follows as such vertices will eventually be a terminals in some decedent of (R, S) in $\mathcal{T}$, and thus by induction will be part of $H_R^{(t)}$.

Induction Hypothesis. *Consider an instance (R, T) at height ℓ in $\mathcal{T}$, then at any time $t \geq t_R$. Then for every $u, v \in \hat{S}_i^{(t)} \cup U_{R_i}^{(t)}$, $d_{\hat{R}^{(t)}}(u, v) \leq d_{H_R^{(t)}}(u, v) \leq (1 + O(\varepsilon))^{\ell-1} \cdot d_{\hat{R}^{(t)}}(u, v)$.*

We will assume that the hypothesis holds in $H_{R_i}^{(t)}$ (for $i \in \{1, 2\}$) w.r.t. set $\hat{S}_i^{(t)} \cup U_{R_i}^{(t)} = \hat{S}_i^{(t)} \cup (U_R^{(t)} \cap \hat{R}_i^{(t)})$, and stretch $(1+O(\varepsilon))^{\ell-1}$, and prove it for $H_R^{(t)}$. Consider a pair of vertices $u, v \in \hat{S}^{(t)} \cup U_R^{(t)}$, and let $Q_{u,v}$ be a shortest $u - v$ path in $\hat{R}^{(t)}$. We can assume that $Q_{u,v}$ does not contain any vertices from $\hat{S}^{(t)} \cup U_R^{(t)}$ (other than u, v). Otherwise, if $Q_{u,v}$ contains another vertex $z \in \hat{S}^{(t)} \cup U_R^{(t)}$, then by induction on the number of hops in $Q_{u,v}$ (that is number of edges), it holds that

$$\begin{aligned} d_{H_R^{(t)}}(u, v) &\leq d_{H_R^{(t)}}(u, z) + d_{H_R^{(t)}}(z, v) \\ &\leq (1 + O(\varepsilon))^\ell \cdot \left(d_{\hat{R}^{(t)}}(u, z) + d_{\hat{R}^{(t)}}(z, v)\right) \\ &= (1 + O(\varepsilon))^\ell \cdot d_{\hat{R}^{(t)}}(u, v). \end{aligned}$$

Suppose first that $Q_{u,v}$ contains an edge e that been added to $\hat{R}^{(t)}$ such that R was a minimal valid instance. The endpoints of e belong to $U_R^{(t)}$. As we assumed that u, v are the only vertices on $Q_{u,v}$ that can belong to $U_R^{(t)}$, it follows that $Q_{u,v}$ is simply the edge $e = \{u, v\}$. Note that in this case $e \in H_R^{(t)}$, and thus $d_{H_R^{(t)}}(u, v) = d_{\hat{R}^{(t)}}(u, v)$, and we are done. We thus can assume that $Q_{u,v}$ does not contain any edges not in $R = \hat{R}^{(t_R)}$.

If the path $Q_{u,v}$ is fully contained in either $\hat{R}_i^{(t)}$ (for $i \in \{1, 2\}$), then using the induction hypothesis $d_{H_R^{(t)}}(u, v) \leq d_{H_{R_i}^{(t)}}(u, v) = (1 + O(\varepsilon))^{\ell-1} \cdot d_{R_i^{(t)}}(u, v) = (1 + O(\varepsilon))^{\ell-1} \cdot d_{\hat{R}^{(t)}}(u, v)$. We thus can assume that $Q_{u,v}$ is not fully contained in either $\hat{R}_1^{(t)}, \hat{R}_2^{(t)}$.

The path $Q_{u,v}$ can be broken into consecutive sub-paths $Q_1, Q_2, \ldots, Q_m$, where Q_j is a path from x_{j-1} to x_j, and is a maximal internal path in $\hat{R}_{i_j}^{(t)}$ (for $i_j \in \{1, 2\}$). The path is maximal in the sense that the next edge along $Q_{u,v}$ after x_j does not belong to $\hat{R}_{i_j}^{(t)}$. See Figure 5 for illustration. Note that the values of $\{i_j\}_{j=1}^m$ are alternating between odd and even indices. As we assume that $Q_{u,v}$ is fully contained in $R = \hat{R}^{(t_R)}$, it must be the case that all of $x_1, x_2, \ldots, x_{m-1}$ lay on π. There are two cases to consider:

- Sub-path Q_j for $j \in \{1, m\}$. Consider Q_1, which is a path from $x_0 \in \hat{S}^{(t)} \cup U_R^{(t)}$ to x_1 which lays on π. Q_1 is internal to $\hat{R}_{i_1}^{(t)}$. In this case, according to the invariant, there is a portal $y_1 \in P_\pi^{(t)}$ such that $d_{\hat{R}^{(t)}[\pi]}(y_1, x_1) \leq \varepsilon \cdot d_{\hat{R}^{(t)}}(x_0, x_1) = \varepsilon \cdot d_{\hat{R}_{i_1}^{(t)}}(x_0, x_1)$. Using the induction hypothesis,

$$\begin{aligned} &d_{H_R^{(t)}}(x_0, y_1) \leq d_{H_{R_{i_1}}^{(t)}}(x_0, y_1) \\ &\leq (1 + O(\varepsilon))^{\ell-1} \cdot d_{R_{i_1}^{(t)}}(x_0, y_1) \\ &\leq (1 + O(\varepsilon))^{\ell-1} \cdot \left(d_{\hat{R}^{(t)}}(x_0, x_1) + d_{R[\pi]}(x_1, y_1)\right) . \end{aligned}$$

  Similarly, we can find a portal $\tilde{y}_{m-1} \in P_\pi^{(t)}$ such that $d_{\hat{R}^{(t)}[\pi]}(\tilde{y}_{m-1}, x_{m-1}) \leq \varepsilon \cdot d_{\hat{R}_{i_m}^{(t)}}(x_{m-1}, x_m)$ and $d_{H_R^{(t)}}(\tilde{y}_{m-1}, x_m) \leq (1 + O(\varepsilon))^{\ell-1} \cdot \Big(d_{\hat{R}^{(t)}}(x_{m-1}, x_m) + d_{\hat{R}^{(t)}[\pi]}(\tilde{y}_{m-1}, x_{m-1})\Big)$.
- Sub-path Q_j for $j \in \{2, \ldots, m-1\}$ (if any). Here both endpoints x_{j-1}, x_j lay on π. As we assumed that $Q_{u,v}$ is fully contained in $R = \hat{R}^{(t_R)}$, it cannot be the case that Q_j is simply the consecutive path from x_{j-1} to x_j along π. This is, as otherwise Q_{j-1} would not have been a maximal internal path (as $Q_{j-1} \circ Q_j$ would've been longer internal path). As π is a shortest path in R, and Q_j is fully contained in R, it must be the case that $w_R(Q_j) \geq d_{R[\pi]}(x_{j-1}, x_j)$. As Q_j is a shortest path in $\hat{R}^{(t)}$, some edges along the sub-path of π from x_{j-1} to x_j have been deleted in $\hat{R}^{(t)}$. Let $e_j = \{z_j, z_j'\}$ be the closest (to x_j w.r.t. π) deleted edge on the sub-path of π from x_j to x_{j-1}. Let z_j be the closer (w.r.t. π) endpoint to x_j. Note that $z_j \in U_R^{(t)}$, and that the sub-path of π from z_j to x_j is fully contained in $\hat{R}^{(t)}$. By the invariant, there is a portal $y_j \in P_\pi^{(t)}$ such that

$$\begin{aligned} d_{\hat{R}^{(t)}[\pi]}(x_j, y_j) \leq \varepsilon \cdot d_{\hat{R}^{(t)}}(x_j, z_j) \leq \varepsilon \cdot d_{\hat{R}^{(t)}[\pi]}(x_j, z_j) \\ le\varepsilon \cdot d_{R[\pi]}(x_{j-1}, x_j) \leq \varepsilon \cdot d_{\hat{R}^{(t)}}(x_{j-1}, x_j) \end{aligned}$$

  Similarly, we can argue that there is a portal $\tilde{y}_{j-1} \in P_\pi^{(t)}$ such that $d_{\hat{R}^{(t)}[\pi]}(x_{j-1}, \tilde{y}_{j-1}) \leq \varepsilon \cdot d_{\hat{R}^{(t)}}(x_{j-1}, x_j)$. Using the induction hypothesis it holds that

$$\begin{aligned} &d_{H_R^{(t)}}(\tilde{y}_{j-1}, y_j) \leq d_{H_{R_{i_j}}^{(t)}}(\tilde{y}_{j-1}, y_j) \\ &\leq (1 + O(\varepsilon))^{\ell-1} \cdot d_{\hat{R}_{i_j}^{(t)}}(\tilde{y}_{j-1}, y_j) \\ &\leq (1 + O(\varepsilon))^{\ell-1} \cdot \Big(d_{\hat{R}^{(t)}[\pi]}(\tilde{y}_{j-1}, x_{j-1}) + d_{\hat{R}_{i_j}^{(t)}}(x_{j-1}, x_j) \\ &+ d_{\hat{R}^{(t)}[\pi]}(x_j, y_j)\Big). \end{aligned}$$

Concatenating all the obtain paths in the emulator we conclude

$$\begin{aligned} &d_{H_R^{(t)}}(u, v) \leq d_{H_R^{(t)}}(x_0, y_1) \\ &+ \sum_{j=2}^{m-1} \Big(d_{H_R^{(t)}}(y_{j-1}, \tilde{y}_{j-1}) + d_{\hat{R}^{(t)}[\pi]}(\tilde{y}_{j-1}, y_j)\Big) \\ &+ d_{\hat{R}^{(t)}[\pi]}(\tilde{y}_{m-1}, y_{m-1}) + d_{H_R^{(t)}}(y_{m-1}, x_m) \\ &\leq (1 + O(\varepsilon))^{\ell-1} \Bigg(\sum_{j=2}^{m} d_{\hat{R}^{(t)}}(x_{j-1}, x_j) \\ &+ O(\varepsilon) \cdot \sum_{j=2}^{m} d_{\hat{R}^{(t)}}(x_{j-1}, x_j)\Bigg) \end{aligned}$$

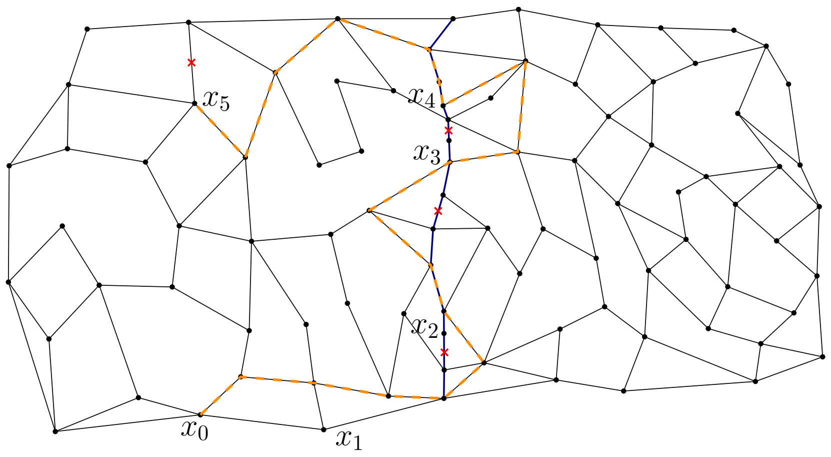


Fig. 5: Illustration of the stretch argument. The red x denote deleted edges. The path $Q_{u,v}$, which is illustrated by a dashed orange line, is the shortest path in $\hat{R}^{(t)}$ from $u = x_0$ to $v = x_5$. Here $x_0 \in \hat{S}_i^{(t)}$ is a portal, while $x_5 \in U_{R_i}^{(t)}$ is incident to an updated edge. We assume that no other vertex along $Q_{u,v}$ belong to this union $\hat{S}_i^{(t)} \cup U_{R_i}^{(t)}$. We partition $Q_{u,v}$ into maximal sub-paths such that Q_1, Q_3, Q_5 are contained in $\hat{R}_1^{(t)}$, and Q_2, Q_4 are contained in $\hat{R}_1^{(t)}$. We first use the invariant to show that we can route through portals on the separator path π, and then use the induction hypothesis to show that the stretch of the emulator between consecutive portals is small.

$$= (1+O(\varepsilon))^{\ell} \cdot \sum_{j=2}^{m} d_{\hat{R}^{(t)}}(x_{j-1}, x_j)$$
$$= (1+O(\varepsilon))^{\ell} \cdot d_{\hat{R}^{(t)}}(x_0, x_m) \ .$$

where the second inequality holds as $d_{\hat{R}^{(t)}[\pi]}(y_j, \tilde{y}_j) \leq d_{\hat{R}^{(t)}[\pi]}(y_j, x_j) + d_{\hat{R}^{(t)}[\pi]}(x_j, \tilde{y}_j) \leq \varepsilon \cdot \left(d_{\hat{R}^{(t)}}(x_{j-1}, x_j) + d_{\hat{R}^{(t)}}(x_j, x_{j+1})\right)$.

This finishes the stretch argument for Type-2 instances. The analysis of Type-3 instances is somewhat similar and we will skip it. The proof for a Type-4 instance $(\hat{R}^{(t)}, \hat{S}^{(t)})$ is straightforward and much easier. Here $(\hat{R}^{(t)}, \hat{S}^{(t)})$ is partitioned into instances $(\hat{R}_1^{(t)}, , \hat{S}_1^{(t)}), \ldots, (\hat{R}_q^{(t)}, , \hat{S}_q^{(t)})$ where all the boundary vertices of the children are portals in $(\hat{R}^{(t)}, \hat{S}^{(t)})$, and terminals in the respective children. This property also holds after arbitrary updates. Consider a pair of terminals $u, v \in S$, with shortest path $Q_{u,v}$. If $Q_{u,v}$ is internal to some $(\hat{R}_i^{(t)}, \hat{S}_i^{(t)})$, then the stretch of u, v is guaranteed by induction. Otherwise, the path $Q_{u,v}$ can be broken into sub-paths $Q_1, Q_2, \ldots, Q_m$ where each Q_j is internal to some instance $(\hat{R}_i^{(t)}, \hat{S}_i^{(t)})$. Note that the endpoints of each such sub-path Q_j are in $\hat{S}_i^{(t)}$, and thus the stretch between the endpoints of Q_j is guaranteed by induction. Overall, we can concatenate all the obtained sub-paths and bound the stretch. Note that actually the stretch does not increased here.

III. Preliminaries

We omit technical proofs of the theorems, lemmas and claims from the conference version of the paper. Interested readers can find the detailed analysis in the full version of this paper.

a) Graphs.: For any graph H, we denote by $E(H)$ the edge set of H, let $V(H)$ be the vertex set of H which we always take to consist of the endpoints of the edges, and let w_H be the weight function of H, that maps each edge in H to a positive real.

In this paper, we work with multigraphs, where each edge is identified by its two endpoints and a unique identifier. However, we sometimes abuse notation and write $e = \{u, v\}$ for an edge e, where u, v are the endpoints of e omitting the identifier. We also sometimes write $e \in H$ to imply $e \in E(H)$ and $v \in H$ to imply $v \in V(H)$ when the context is clear.

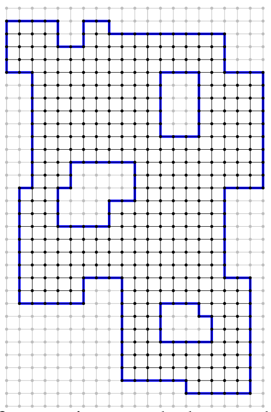
We assume all (multi-)graphs in the rest of this paper to be planar and assume all weights to be reals in $[1, n^4]$ (via a standard reduction which multiplies the runtime by $O(\log W)$, we can handle arbitrary weights in $[1, W]$). We let $H' \subseteq H$ be a subgraph of H meaning that $E(H') \subseteq E(H)$. For graph H and $E' \subseteq E(H)$ and $V' \subseteq V$, we denote by $H[E']$ the edge-induced graph and by $H[V']$ the vertex-induced graph, both with the obvious underlying embedding obtained from the embedding of H.

b) Distances And Shortest Paths.: For any graph H, we denote the distance between vertices $u, v \in V(H)$ by $\texttt{dist}_H(u, v)$. From [26], it is well-known that one can compute/maintain a set of extended weights over paths in H such that taking the lexicographically shortest path π_{uv} from u to v in H with respect to the real weight of the path and then the extended weight ensures uniqueness of π_{uv} and preserves the subpath property, i.e. for every vertices $x, y \in \pi_{uv}$, we have that $\pi_{uv}[x, y]$ is the lexicographically shortest path from x to y.

c) Embeddings.: Throughout the paper, we assume that each planar graph is given with a fixed embedding into the plane. More precisely, our algorithm implicitly maintains a combinatorial embedding (also referred to as a rotation system).

To define a combinatorial embedding of a graph H, we introduce the notion of *darts* $E(H)\times\{\pm 1\}$. Assign the vertices arbitrary distinct labels from the positive integers. We use e^+ and e^- as short-hands for $(e,+1)$ and $(e,-1)$ for edge $e=\{u,v\}\in E(H)$, define $\mathit{rev}(e^+)=e^-$ and $\mathit{rev}(e^-)=e^+$ and define the tail $\mathit{tail}(e^+)$ to be the vertex among u,v with smaller label and by head $\mathit{head}(e^+)$ the vertex with a larger label. We define $\mathit{tail}(e^-)=\mathit{head}(e^+)$ and $\mathit{head}(e^-)=\mathit{tail}(e^+)$. We say u and v are the *endpoints* of e,e^+ and e^-. We sometimes write uv to denote the dart $d\in\{e^+,e^-\}$ that is oriented from vertex u to v, i.e. $u=\mathit{tail}(d)$ and $v=\mathit{head}(d)$ (this again abuses notation slightly since we allow for multi-edges).

A combinatorial embedding $\mathcal{E}_H$ is a permutation of the set of darts whose orbits have a one-to-one mapping with V. More precisely, for each orbit associated with $v\in V(H)$, the restriction of $\mathcal{E}_H$ to that orbit, denoted $\mathcal{E}_H|v$, is a permutation cycle. The permutation cycle for v specifies how the darts with head v are arranged around v in the embedding (in, say, counter-clockwise order).

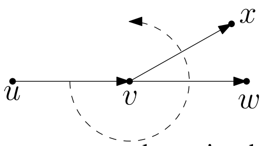


d) Holes.: Given a graph H and a connected subgraph $H'\subseteq H$. We call a face f in the embedding of H' a *hole* with respect to H if f is not a face in H. We denote by $h(H',H)$ the collection of holes of H' with respect to H. Note that we only talk about holes when considering connected subgraphs of H' of H, so that each hole is a cycle. We say that H' is an *h-hole graph* w.r.t. H if H' has at most h holes w.r.t H. We often write one-hole graph in lieu of 1-hole graph. See the illustration on the right of a 4-hole graph H'. The edges of H' that are incident to a hole are colored blue while the edges of H' that are not are colored black. Edges in H but not in H' are colored grey.

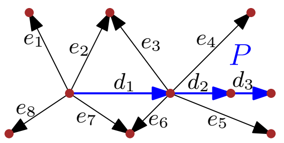
For $h\in h(H',H)$, we denote by $H\setminus h$ the graph H after removing all edges and vertices that are in (the image of) h. This generalizes straightforwardly to define $H\setminus\{h_1,h_2,\dots,h_k\}=((((H\setminus h_1)\setminus h_2)\dots)\setminus h_k)$. We define the image of a graph H' w.r.t H to be the region $H\setminus h(H',H)$.

e) Paths and Cycles.: A path P is an ordered alternating collection of vertices and darts $\langle v_0,d_0,v_1,d_1,\dots,v_k,d_k,v_{k+1}\rangle$ where for each $0\le i\le k$, we have $v_i=\mathit{tail}(d_i), v_{i+1}=\mathit{head}(d_i)$. We say that $P=\langle v_0,d_0,v_1,d_1,\dots,v_k,d_k,v_{k+1}\rangle$ is a *cycle* if $v_0=v_{k+1}$. We define $\mathit{rev}(P)=\langle v_{k+1},\mathit{rev}(d_k),v_k,\dots,v_1,\mathit{rev}(d_0),v_0\rangle$, $\mathit{tail}(P)=v_0$ and $\mathit{head}(P)=v_{k+1}$. For P,P' with $\mathit{head}(P)=\mathit{tail}(P')$, we let $P\circ P'$ be the concatenation of P and P'. If P is a path with k darts, we let $P[i,j]$ for $0\le i\le j\le k$ denote the *segment* of P from the i-th vertex to its j-th vertex. We say a segment $P[i,j]$ is *internal* to P if $0<i\le j<k$. For a cycle C, we also allow for $j\le i$, with the obvious meaning of segment $C[i,j]$ in this case.

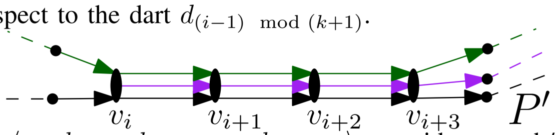
While our notation for paths is non-standard, it is necessary to formally distinguish multi-edges and to allow for trivial segments consisting only of a single vertex. We assume in this article that all paths, however, are non-trivial, only segments might not be. We often abuse notation and write either $P=\langle d_0,d_1,\dots,d_k\rangle$ or $P=\langle v_0,v_1,\dots,v_{k+1}\rangle$ when the context is clear. We say that a path P is *simple* if no vertex appears twice on the path, and say that a cycle P is *simple* if all but the first/last vertex appear only once and the first/last vertex appears exactly twice.



f) Left and Right Order.: Consider a vertex $v\in V(H)$ with dart uv, and darts vw and vx. We say that vx is to the *left* of vw w.r.t. uv if xv occurs strictly between wv and uv in the counter-clockwise order of elements in the orbit $\mathcal{E}|v$. See the illustration to the right. Similarly, vx is to the *right* of vw w.r.t. uv if xv occurs strictly between uv and wv in the counter clockwise order.



For any simple path $P=\langle d_0,d_1,\dots,d_k\rangle$ in H, and a dart $\overline{d}$ in H with $\mathit{tail}(\overline{d})=\mathit{tail}(d_i)$ for some $0\le i\le k$, then we say that $\overline{d}$ emanates *left* (*right*) of P if $i=0$ or if $\overline{d}$ is to the left (right) of the dart d_i with respect to the dart d_{i-1}. For example, in the figure on the right, e_3 and e_4 emanate to the left of P, e_5 and e_6 emanate to the right of P, while e_1,e_2,e_7,e_8 emanate both to the right and to the left of P. If $P=\langle d_1,d_2,\dots,d_k\rangle$ is a simple cycle, we say dart $\overline{d}$ in H, with $\mathit{tail}(\overline{d})=\mathit{tail}(d_i)$ for some $1\le i\le k$, emanates *left* (*right*) of P if $\overline{d}$ is to the left (right) of the dart d_i with respect to the dart $d_{(i-1)\bmod (k+1)}$.



Again for simple path $P=\langle v_0,d_0,v_1,d_1,\dots,v_k,d_k,v_{k+1}\rangle$, consider a path/cycle P' such that $P[i,j]$ is a segment of P that also appears on P'. We say that $P[i,j]$ is *left* (*right*) of P' if either

- $i=j$ and therefore $P[i,j]$ is the empty path containing only a single vertex, or
- the i-th vertex is the tail of P or the tail of P' and P' is not a cycle, and the $(j-1)$-th dart d is left (right) of P' (like the purple path on the right), or
- for both the $(i-2)$-th dart d' and the $(j-1)$-th dart d, we have $\mathit{rev}(d')$ and d are left (right) of P' (like the green path on the right).

Note that we treat the special case where the segment is trivial separately and can therefore use our definitions for non-trivial paths.

Definition III.1 (Crossing Paths)**.** *We say for any simple paths P,P' that P is* crossing *P' if one of the following occurs:*

- *There is a segment of $P[i,j]$ of P with $P[i,j]\subseteq P'$ that is neither left nor right to P'.*
- *if there is a segment of $P[i,j]$ of P with $P[i,j]\subseteq \mathit{rev}(P')$ that is neither left nor right to $\mathit{rev}(P')$.*
- *if the endpoints of P appear as internal vertices on P' or vice versa.*

Since the relation is symmetric, we also say that P, P' are crossing; otherwise, say that they are non-crossing.

g) Path-Collections with Planarity-Preserving Properties under Contraction.: It is not difficult to verify the following fact.

Fact III.2 (Slicing Along Path)**.** *Given the graph H and a path P. Let H' be the graph obtained by* slicing *along the path P by duplicating each edge (associated with a dart) and internal vertex on P, and let P' be this copy of P. We then can find a planar embedding of H' where the first dart of P' appears just after the first dart of P in the orbit of $\mathcal{E}$ around the first vertex of P and P', and each edge with a dart appearing to the left of P in H has its endpoint re-mapped to the corresponding copied vertex on P'. Thus $P \circ rev(P')$ forms a face in this new embedding. See Figure 6 for illustration. This embedding of H' can be computed in time $O(|P|)$.*

Given a collection of simple paths $\mathcal{P}$ in H, we let $\mathcal{M}_H[\mathcal{P}]$ be the graph over the endpoints of all paths in $\mathcal{P}$, denoted $\mathcal{V}(\mathcal{P})$, where the edge set consists of an edge $e = \{tail(P), head(P)\}$ for each path $P \in \mathcal{P}$ of weight $w_H(P)$. Next, we prove the following crucial fact.

Fact III.3. *Consider a planar graph H, and let $\mathcal{P}$ be a collection of simple, pairwise non-crossing paths in H. Then $\mathcal{M}_H[\mathcal{P}]$ is a planar graph.*

h) Instances.: A planar, undirected graph H and a subset of vertices $T \subseteq V(H)$ also referred to as *terminals* form an *instance* (H, T) to our algorithm. Given a graph H and an instance (H', T) with $H' \subseteq H$, we say that (H', T) is an h-hole instance w.r.t. H if H' is an h-hole graph w.r.t. H **and** every terminal in T is incident to a hole.

i) Dynamic Graphs.: The focus of this paper is on dynamic graphs, that is a graph H that undergoes a sequence of (batches of) edge insertions/ deletions. We also implicitly allow for vertex insertions/ deletions by removing a vertex from a graph if it is isolated and by allowing insertions of edges with endpoints not yet in the graph. We assume that an edge that is deleted at some point is not added again to the graph at a later point (since we allow for multi-edges this still allows for inserting an edge between the same endpoints, however, the edge has to have a different identifier). We denote by $H^{(t)}$ the graph H after the first t update batches have been applied to H and similarly, we let $\mathcal{E}^{(t)}$ denote the corresponding rotation system after t update batches. In particular, $H^{(0)}$ is the initial graph and $\mathcal{E}^{(0)}$ the initial embedding. More generally, we let $x^{(t)}$ be the value of any variable x just after the algorithm completes processing the t-th update batch to H.

In this paper, we only allow for updates that ensure that the resulting graph is still connected, respect the current embedding of H and require each update to encode not only the changes to the current H but also the changes to the rotation system $\mathcal{E}$. Here, we enforce that the graph is connected in order to keep the embedding unambiguous.

More precisely, initially, $H^{(0)}$ has an embedding in the plane represented by the rotation system $\mathcal{E}^{(0)}$. Consider now the t-th update batch, assuming inductively that $H^{(t-1)}$ has an embedding in the plane represented by the rotation system $\mathcal{E}^{(t-1)}$ are valid graph and embedding. Consider first that the t-th update batch consists of exactly one update. Then, if the update is a deletion of an edge e, the encoding of the update consists of $\{e, \mathcal{E}^{(t-1)}(e, -1), \mathcal{E}^{(t-1)}(e, +1)\}$ which allows to quickly delete the edge e from $H^{(t-1)}$ and to remove the darts $(e, -1)$ and $(e, +1)$ from their orbits. Inserting an edge e can be encoded by $\{e, d^-, d^+\}$ where d^- is the dart such that $(e, -1)$ is to the right of d^- in the new embedding, and $(e, +1)$ is to the right of d^+ (if the vertex did not exist before and is therefore incident only to e then it is straight-forward to add the darts).

We note that from our description of the update (batch) encodings, the encoding of each update takes $O(1)$ words of space, and we can additionally maintain the graph H and its rotation system with worst-case update and query time $O(1)$ (and initialization time $O(n)$).

j) Emulators.: In this paper, we define emulators as follows.

Definition III.4. *Given an instance (G, T). We say that a collection of (pairwise) non-crossing paths $\mathcal{P}$ in G induces an ϵ-emulator $(M_H[\mathcal{P}], T)$ of (G, T) if $T \subseteq V(\mathcal{P})$ and for any $u, v \in T$*

$$\mathtt{dist}_G(u, v) \leq \mathtt{dist}_{M_H[\mathcal{P}]}(u, v) \leq (1 + \varepsilon) \cdot \mathtt{dist}_G(u, v).$$

We use the following crucial theorem that appears in [53].

Theorem III.5 ([40], [53])**.** *Given an instance (G, T), there is a deterministic algorithm* QUARTICEMULATOR(G, T) *that runs in time $O(|T|n)$ and returns a collection of $O(|T|^4)$ internally-disjoint paths $\mathcal{P}$ that induce an 0-emulator $(M_H[\mathcal{P}], T)$.*

k) Dynamic Instance.: Given a tuple (H, T) where H is a dynamic graph and T is a set of vertices that undergoes batches of insertions/ deletions of vertices such that at any time $T \subseteq V(H)$, then we say that (H, T) is a dynamic instance. We define $(H, T)^{(t)} = (H^{(t)}, T^{(t)})$. Again, we let $x^{(t)}$ be the value of any variable x just after the algorithm completed processing the t-th update batch to (H, T).

Theorem III.6 (see [42])**.** *Given a graph G, there is a procedure* FINDBRIDGES(G) *that returns the set of all bridges in G in time $O(n + m)$.*

l) Divisions and Separators.: For any set $X \subseteq V(H)$, we denote by $\partial_H X$ the set of vertices in X that have a neighbor in $V(H) \setminus X$ in H. We often write ∂X in place of $\partial_H X$ when the context is clear, and refer to it as the *boundary* of X. Using a classic result by Lipton, we can recursively partition our input graph while balancing different parameters. Here, we state a result implicitly obtained by [50].

Theorem III.7 (see [50])**.** *Given a simple planar graph G and a subgraph $H \subseteq G$ with holes $h_0, h_1, \ldots, h_\ell$ with respect*

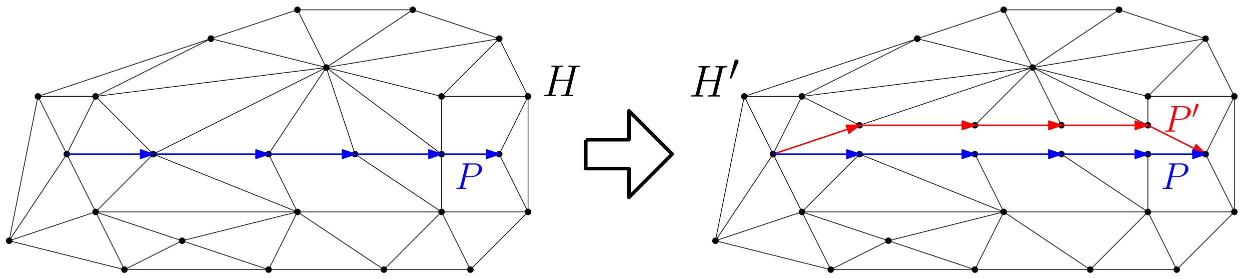


Fig. 6: Illustration of the slicing operation.

to G. Then, there is an algorithm PARTITION(G, H) *that returns $O(1)$ edge induced subgraphs $H_0 = H[E_0], H_1 = H[E_1], \ldots, H_k = H[E_k]$ where $E_1, E_2, \ldots, E_k$ partition the edge set $E(H)$ such that for each $0 \leq i \leq k$, we have*

- *$|V(H_i)| \leq \frac{3}{4}|V(H)| + C_{Boundary} \cdot \sqrt{|V(H)|}$, and*
- *$|\partial_G H_i| \leq \frac{3}{4}|\partial_G H| + C_{Boundary} \cdot \sqrt{|V(H)|}$ for some small constant $C_{Boundary}$, and*
- *H_i is incident to at most $\frac{3}{4}\ell + 4$ holes with respect to G.*

Additionally, it guarantees that $\sum_i |\partial_G H_i| \leq |\partial_G H| + C_{Boundary} \cdot \sqrt{|V(H)|}$. The algorithm runs in time $O(|V(H)|)$.

Additionally, we also use shortest-path separators. Here, we generalize a notion introduced in [15].

Definition III.8. *Given a graph G, we say an edge $e \in E(G)$ is a bridge if the number of connected components in $G \setminus e$ is strictly larger than the number of connected components G.*

Definition III.9. *Given an instance (G, T) where G is connected. Then, we say that a pair $u, v \in V(f_\infty(G))$ is c-balanced for $\frac{1}{2} \leq c < 1$, if each connected component of $f_\infty(G) \setminus \{u, v\}$ contains at most a c-fraction of the vertices in T. We also say that the (lexicographically) shortest uv-path π_{uv} is a c-balanced shortest path separator.*

Theorem III.10. *Given an instance (G, T) where G is an n-vertex graph, there is an $O(n)$ time algorithm that finds a $\frac{11}{12}$-balanced pair $u, v \in V(f_\infty(G))$ such that at least one of the following holds:*

1) *the lexicographically shortest uv-path π_{uv} does not contain a bridge edge of G.*
2) *there is an edge $e = (u, v) \in E(G)$ such that e is a bridge.*

m) Covers.: In this paper, we use multiplicative ε-Path-Covers.

Definition III.11 (Multiplicative ε-Cover.). *Given an instance (G, T) and a shortest uv-path π_{uv} for vertices $u, v \in V$. We say that a set of vertices $C \subseteq V(\pi_{uv})$ is an ε-cover for T if for any vertices $t \in T, p \in \pi_{uv} \setminus \{t\}$, we have that $\min_{c \in C} \mathtt{dist}_{\pi_{uv}}(c, p) \leq \varepsilon \cdot \mathtt{dist}_G(t, p)$.*

In other words, an ε cover of T is a set of vertices C, such that for every vertices $t \in T, p \in \pi_{uv} \setminus \{t\}$, there is a portal $C \in C$ on $\pi_{u,v}$ at distance $\varepsilon \cdot \mathtt{dist}_G(t, c)$ from p. Note that the shortest path from T to p that goes through c has weight at most $\mathtt{dist}_G(p, t) + 2 \cdot \mathtt{dist}_G(p, c) \leq (1 + 2\varepsilon) \cdot \mathtt{dist}_G(t, p)$.

Lemma III.12 (see Section 3.1, [48]). *Given an instance (G, T) and a (lexicographically) shortest uv-path π_{uv} for vertices $u, v \in V$, and some parameter $\epsilon > 0$. Then, there is an algorithm* SOURCEPATHPORTALS$(G, \pi_{uv}, T, \epsilon)$ *that returns an ε-cover for T of size at most $O(|T|/\epsilon)$. The algorithm runs in time $O(n|T|)$.*

Given a one-hole instance (R, S), we define PORTALSALONGPATH(R, π_{uv}, δ, L) procedure of computing *portal* set along the shortest path between c-balanced pair of vertices $u, v \in V(f_\infty)$ with parameters δ, L from [15] which can be described as follows: for each $i = 1, 2, \ldots, L$

1) Add $\operatorname{argmin}_{w \in \pi_{uv}} \mathtt{dist}_{\pi_{uv}}(x, w) \leq \exp(-i \cdot \delta)$ to the portal set, for $x \in \{u, v\}$,
2) Add $\operatorname{argmax}_{w \in \pi_{uv}} \mathtt{dist}_{\pi_{uv}}(x, w) \geq \exp(-i \cdot \delta)$ to the portal set, for $x \in \{u, v\}$.

The total number of portals in the path separator π_{uv} is bounded by $C \cdot \log n/\delta$ for some constant $C > 0$ since the length of any path is bounded by $O(n^5)$ because all the edge weights belong to $[1, n^4]$.

Due to [15], let $T =$ PORTALSALONGPATH$(R, \pi_{uv}, S \cup \{u, v\}, \epsilon)$ for graph R such that S lies on the outer face of R and π_{uv} c-balanced separator of R w.r.t. S then T induces an ϵ cover for S on path π_{uv} such that $|T| = \frac{C}{\epsilon} \cdot \log n$.

IV. An Emulator Hierarchy

The main technical result of this article is the following theorem that allows us to maintain a much smaller emulator of an instance with very small recourse.

Theorem IV.1. *Given a dynamic instance (G, T), a size parameter n that upper bounds the number of vertices in G at any time and with G having all its weights at all times in $[1, n^4]$, the guarantee that T is of size at most 2 at any time, a target parameter $1 \leq \tau \leq \frac{n}{\log(n)}$, and a precision parameter $0 < \epsilon < 1$.*

Then, there is a deterministic algorithm that maintains explicitly a collection of paths $\mathcal{P}$ in G such that $\mathcal{P}$ induces an ϵ-emulator $(\mathcal{M}_G[\mathcal{P}], T)$ of (G, T) such that

- *Emulator Size: at any time, $|\mathcal{P}| = \tilde{O}(\sqrt{n\tau}/\epsilon^2)$, and*
- *Emulator Recourse: each update batch B to (G, T) can be processed such that the amortized number of paths deleted and added to $\mathcal{P}$ is at most $\tilde{O}(|B|/\epsilon^2)$, and*

- *Update Time: each update batch B to (G,T) can be processed in amortized time $\tilde{O}\left(|B|\frac{n}{\epsilon^2\tau}\right)$.*

The algorithm takes initialization time $\tilde{O}(n/\epsilon^2)$.

To obtain Theorem I.1, we will only require the terminal set T to consist of two vertices $\{s,t\}$ at any given time in order to run queries for the distance from s to t on the much smaller graph $\mathcal{M}_G[\mathcal{P}]$. While we could set τ to be very small, even near-constant, which makes this graph very small as well, this would result in an enormous update time of $\Omega(n)$ per update. We, therefore, set τ much larger and run multiple instances of Theorem IV.1 recursively on each other to reduce the size of the emulator.

We defer the proof sketch of Theorem IV.1 to Section V and complete this section by sketching the proof of Theorem I.1 precise.

Henceforth, we let c_{size} denote the hidden polylogarithmic factor in the emulator size bound stated by Theorem IV.1, i.e. we have at all times that the path collection $\mathcal{P}$ that is maintained is of size at most $c_{size} \cdot \sqrt{n\tau}/\epsilon^2$. We analogously define c_{rec} to be the hidden polylogarithmic factor in the emulator recourse stated by Theorem IV.1, i.e. we have that processing any batch B of updates to the current instance results in at most $c_{rec} \cdot |B|/\epsilon^2$ recourse in the number of paths $\mathcal{P}$ and thus emulator $\mathcal{M}_G[\mathcal{P}]$.

a) An Emulator Hierarchy.: Let us now describe our emulator hierarchy. We let $k = \log^{2/3} n$ (where we assume w.l.o.g. that k is integer), and define $\delta = 2^{\log^{2/3} n}$. In our algorithm, we denote by (G,T), where T is initially empty, the instance obtained from interleaving the update sequence of G such that whenever the adversary issues a query for the approximate distance between two vertices $s,t \in V(G)$, the instance (G,T) simulates this query by adding s and t to the set T, then extracts the distance estimate and finally removes s and t from T again.

Given this encoding of the updates, we let $(H_1 = G,T),(H_2,T),\ldots,(H_k,T)$ be a hierarchy of emulators that are maintained by our algorithm such that for each $1 \leq i < k$, we let the instance $(H_{i+1} = \mathcal{M}_{H_i}[\mathcal{P}_i],T)$ be the emulator produced by running the algorithm from Theorem IV.1 on the instance (H_i,T) with size parameter $n_{H_i} = n \cdot \left(\frac{2c_{size}}{\epsilon^2\sqrt{\delta}}\right)^{i-1}$, target parameter $\tau_i = n_{H_i}/\delta$ and precision parameter ϵ.

Now, whenever the adversary issues a query for the approximate distance between two vertices s,t, we take the graph H_k at the time where (G,T) has G being equal to the current graph and $T = \{s,t\}$. We then run Dijkstra's algorithm from s to find the exact distance from s to t on H_k. We return this distance as the estimate for the distance from s to t in the current graph G.

Given the set of parameters, we complete the proof Theorem I.1 using standard induction argument due to Theorem IV.1. We omit the details of the full analysis here and refer interested readers to the full version of the paper.

V. Emulator Maintenance

In this section, we give the key technical components and algorithms which leads to the proof of Theorem IV.1 which is our main technical contribution.

Here, we focus on the case where each update batch consists of a single update. It is not hard to see that this is without loss of generality. In this section, we focus on the algorithm to maintain the emulator and omit the technical details of the proofs the main theorem due to space constraints. We refer the interested readers to the full version of the paper for detailed technical analysis.

A. High-Level Overview

The key to our algorithm is a decomposition of the input graph G into small pieces. Here, we decompose the graph G recursively into smaller and smaller pieces until every piece is sufficiently small. To keep track of the recursive decomposition, we maintain a decomposition tree $\mathcal{T}$ where each node is associated with a static instance (R,S). As all nodes correspond to instances, we use the words interchangeably. Here by static, we mean that the instance (R,S) is not dynamic but instead refers to the graph R and the vertex set S as it was when the node was added to $\mathcal{T}$. The root node of $\mathcal{T}$ consists of the instance $(G,\emptyset)$ where G is the initial input graph and each internal node $(R,S) \in \mathcal{T}$ has its children being instances whose graphs (roughly) partition R.

Given a decomposition tree $\mathcal{T}$, we then suggest an emulator that is built in a bottom-up fashion from the tree $\mathcal{T}$. We show in fact, that the changes that we make to $\mathcal{T}$ as the graph G evolves over time only cause very few changes to the emulator induced by $\mathcal{T}$. Further, we can efficiently compute all of these changes. This yields an algorithm with polylogarithmic recourse in the emulator and subpolynomial update time.

We next describe how to initialize the decomposition tree $\mathcal{T}$ on the input graph G, or rather the instance $(G,\emptyset)$. We then discuss an update algorithm to maintain $\mathcal{T}$, essentially by cleverly rebuilding subtrees of $\mathcal{T}$. Finally, we define how this decomposition tree induces an emulator.

We then analyze these algorithms in the full version of this paper which turn out to be highly non-trivial.

B. Initialization of the Decomposition Tree

a) Creating the Decomposition Tree.: We initialize our decomposition tree by running Algorithm 1. It creates the rooted decomposition tree $\mathcal{T}$ with a single root node corresponding to the initial input instance (although we remove the terminal set T for the technical reason that the terminals do not appear on holes of the instance since G has not holes w.r.t. itself).

To conclude our initialization procedure, we finally invoke the procedure INITIALIZEINSTANCE$(\cdot)$ on the root node. This procedure recursively constructs the decomposition tree.

b) Initializing Instances (Recursively).: The procedure INITIALIZEINSTANCE$(\cdot)$ implemented by Algorithm 2 is the workhorse behind our algorithm. It takes an instance (R,S) and a time t where R is a subgraph of G at time t. We first

```
Algorithm 1: INITIALIZE()
1: Create a rooted decomposition tree $\mathcal{T}$ with root node $(G, \emptyset)$.
2: INITIALIZEINSTANCE($(G, \emptyset), 0$).
```

give a high-level description of the algorithm and then discuss implementation details.

At a high level, the procedure initializes each instance (R, S) by creating a portal set $\text{PORTALS}_{(R,S)}$ (see Line 9). These portal sets will later help us to gather the necessary information for creating the emulator from $\mathcal{T}$ as they describe for each instance (R, S) the change to S since the creation of the instance (R, S). It then verifies whether (R, S) is an instance such that R and S are very small in which case it determines that (R, S) should become a leaf node in $\mathcal{T}$ and returns (see Line 10). Otherwise, it decomposes (R, S) into smaller instances, adds each of these instances as children of (R, S) to $\mathcal{T}$, and then recursively initializes each of these child instances (see Line 43).

Here we glossed over two major points: firstly, if (R, S) is such that R is not connected, we are in a degenerate case. In this case, we simply find the connected components of R and then remove (R, S) in $\mathcal{T}$ and add instances induced on the connected components as children to (R, S)'s former parent (see the if-statement starting in Algorithm 1). Finally, we initialize each of the newly created instances.

Otherwise, we have that (R, S) has R being a connected graph and thus it is non-degenerate. Thus, we initialize its portal set $\text{PORTALS}_{(R,S)}$ (see Line 9) and set it to be equal to S, the set of terminal vertices on the instance. So far, we have only discussed in detail the case where R and S are both small in which case we decide that (R, S) should become a leaf in $\mathcal{T}$ and thus return (see Line 10). We also say that (R, S) is of Type-1. If (R, S) is not of Type-1, however, we decompose (R, S) further by adding children instances to (R, S) in $\mathcal{T}$. Here, we need to carefully decompose (R, S) to make our algorithm efficient. Therefore, we label instance (R, S) as Type-2,3, or 4; based on this labeling, we then chose a decomposition strategy that produces new instances that are taken to be the children of (R, S) and on which we then again recurse.

In our algorithm, we enforce that along any root-to-leaf path in $\mathcal{T}$, the Type-on the instances is monotonically decreasing. That is the instances higher up in the tree are mostly of Type-4, then we have some instances that are of Type-3, some of Type-2, and finally, all leaves are of Type-1.

c) Decomposing Instances by Type.: Here, we define the Type-4 nodes (see Line 36) such that the decomposition formed by the subtree of $\mathcal{T}$ consisting only of Type-4 nodes roughly corresponds to an r-division for $r = \frac{n}{\tau}$. Recall that an r-division is an edge partition such that each graph in this partition has size at most r, the sum of boundary nodes is at most $O(n/\sqrt{r})$ and each graph in the partition set has at most $O(1)$ holes w.r.t. G.

Once each graph is sufficiently small, i.e. of size $O(n/\tau)$, we label it Type-3 (see Line 27) if it is incident to multiple holes. Type-3 instances (R, S) are then decomposed by finding a shortest path π_{uv} between two vertices on different holes and slicing the graph open along this path by copying the vertices on it to obtain a new graph R' that has at least one hole less than the instance (R, S). Note that we also need to apply the slicing operation to the graph that we compare to when defining holes. Formally, let us make the following definition.

Now, when talking about holes, we always refer to holes w.r.t. graph G. It is not hard to verify that R' has exactly one hole less w.r.t. $G_{(R',S')}$ than R w.r.t. G. Note that G might now contain multiple copies of a single edge in G. However, since we enforce that Type-4 nodes only have $O(1)$ holes, we can establish that this results in at most $O(1)$ copies of each edge in G in any graph G.

Once an instance (R, S) has R of small size, i.e. $O(n/\tau)$, and R is only incident to a single hole, but we still have that S of size larger c_{CutOff}, we label (R, S) as Type-2 (see Line 13). Here our decomposition strategy uses a *balanced* shortest path separator w.r.t. S. The separator ensures that the two child instances created to (R, S) are one-face instances, and the number of terminal vertices in these instances is at most a constant fraction of $|S|$ while also the total number of all terminal vertices in child instances is not much larger than $|S|$. For technical reasons, we cannot allow for the child instances to contain multiple copies of an edge that is a bridge edge in R. However, we can either find a shortest path separator π_{uv} that does not contain a single bridge edge, or we can identify a single bridge edge whose removal yields a good separator.

As previously mentioned, as soon as additionally the size of set S in instance (R, S) drops below c_{CutOff}, we label it Type-1 (see Line 10) and keep it as a leaf.

We note that the above discussion is slightly vague with the exact threshold for the size of R, labeling it Type-4 or below. This is for a technical issue: we want to enforce monotonicity of instance types along root-to-leaf paths in $\mathcal{T}$, i.e. walking along such a path, one might find an instance (R, S) that has size smaller n/τ and be labeled Type-2 or 3 depending on the number of holes. But when such a node (R, S) is further decomposed, its children do not properly partition the graph R but instead, vertices on the shortest path separator might appear twice. This might cause the child instance(s) of (R, S) to have graph size slightly larger than n/τ again. However, as we show later, none of these nodes can ever have size larger than $O(n/\tau)$.

C. Updating Information in the Decomposition Tree

Next, we discuss how to update the decomposition tree $\mathcal{T}$ after every adversarial update to G. Before we can describe our algorithm, we require some basic definitions that formalize how instances (R, S) evolve dynamically (we keep instances (R, S) in $\mathcal{T}$ to be static but we need to maintain a graph $\hat{R}^{(t)}$ that simulates edge updates to G up until time on R if they apply).

Algorithm 2: InitializeInstance$((R,S),t)$

```
1: if $R$ is not a connected graph then
2:     Remove $(R,S)$ from the tree $\mathcal{T}$.
3:     for connected component $C$ in $R$ do
4:         Add instance $(R[C], S \cap C)$ as a child of $(R', S')$ (patent of $(R,S)$) to $\mathcal{T}$.
5:         InitializeInstance$((R[C], S \cap C), t)$.
6:     end for
7:     return
8: end if
9: Portals$_{(R,S)} \leftarrow S$.
10: if $|R| < \frac{n}{\tau}$ and $|S| \le c_{CutOff} \overset{\text{def}}{\coloneqq} \log^{10}(n)$ then
11:     return ,
12: end if
13: if parent instance is of Type-2-3 and $(R,S)$ is a one-hole instance or $|R| < \frac{n}{\tau}$ and $(R,S)$ is a one-hole instance w.r.t. $G_{(R,S)}$ then
14:     Let $u, v \in f_\infty(R)$ be $\frac{11}{12}$-balanced w.r.t. $S$ ( Theorem III.10)
15:     if $\pi_{uv}$ does not contain a bridge edge then
16:         Let $R_1, R_2$ be the (non-empty) regions of $R$ formed by $\pi_{uv}$ and the hole $f_\infty(R)$.
17:         $P \leftarrow$ PortalsAlongPath$(R, \pi_{uv}, S \cup \{u,v\}, \epsilon')$.
18:         Portals$_{(R,S)} \leftarrow$ Portals$_{(R,S)} \cup P$.
19:     end if
20:     if $\pi_{uv}$ is a bridge edge $e$ then
21:         Portals$_{(R,S)} \leftarrow$ Portals$_{(R,S)} \cup \{u,v\}$.
22:         Let $C$ and $C'$ be the connected components of $R \backslash \{e\}$
23:         Let $R_1 \leftarrow R[C] \cup \{e\}$ and $R_2 \leftarrow R[C']$.
24:     end if
25:     Add instances $(R_i,$ Portals$_{(R,S)} \cap R_i)$ for $i = 1,2$ as a child of $(R,S)$ to $\mathcal{T}$.
26: end if
27: if parent instance is of Type-3 or $|R| < \frac{n}{\tau}$ then
28:     Let $h_0, h_1, \ldots, h_\ell$ consist of the holes of $R$ w.r.t. $G_{(R,S)}$.
29:     Let $\pi_{uv}$ be the shortest path between the closest holes $h, h'$with $v \in V(h)$ and $u \in V(h')$.
30:     Portals$_{(R,S)} \leftarrow$ Portals$_{(R,S)} \cup$ SourcePathPortals$(R, \pi_{uv}, S, \frac{\epsilon'}{h})$.
31:     Let $e_u$ be the first edge when walking from $u$ in clock-wise order along $h$;
32:     Let $e_v$ be the first edge when walking from $v$ in anti-clock-wise order along $h'$.
33:     Let $R'$ be the graph obtained by slicing up $R$ along $e_u \oplus \pi_{uv} \oplus e_v$.
34:     Let $S'$ be the Portals$_{(R,S)}$ and all its copies.
35: end if
36: if Otherwise then
37:     Invoke Theorem III.7 on $R$ to obtain the subgraphs $R_0, R_1, \ldots, R_k$.
38:     for $i \in [0,k]$ do
39:         Add the node $(R_i, (S \cup \partial_G R_i) \cap V(R_i))$ to $\mathcal{T}$ as a child of $(R,S)$.
40:     end for
41: end if
42: for child $(R', S')$ of $(R,S)$ in $\mathcal{T}$ do
43:     InitializeInstance$((R', S'), t)$.
44: end for
```

Definition V.1 (Evolving (Sub-)graphs)**.** *Given a static graph R and a dynamic graph H such that at some time t_R, we have $R \subseteq G^{(t)}$. Then, we recursively define for each $t \ge t_R$, the graph $\hat{R}^{(t)}$ as follows:*

- *we let $\hat{R}^{(t_R)} = R$, and*
- *for $t > t_R$, we define $\hat{R}^{(t)}$ to be the graph obtained from applying the t-th update to $G^{(t-1)}$ to the graph $\hat{R}^{(t-1)}$ if either*
  - *the t-th update to $G^{(t-1)}$ is an edge deletion of an edge in $\hat{R}^{(t-1)}$ then we remove the edge and all its copies from $\hat{R}^{(t-1)}$, or*
  - *the t-th update to $G^{(t)}$ is an edge insertion of an edge e and there exists a connected component C in $\hat{R}^{(t-1)}$ and a non-hole face f in $\hat{R}^{(t-1)}[C]$, i.e. a face f that is not in $h(\hat{R}^{(t-1)}[C], G^{(t-1)})$, such that the image of e is contained in the image of f and at least one of the endpoints of e is in C.*

  Otherwise, we let $\hat{R}^{(t)} = \hat{R}^{(t-1)}$. If $\hat{R}^{(t)} \ne \hat{R}^{(t-1)}$, we say that the edge e that is updated in the t-th update is a valid edge update to $\hat{R}^{(t-1)}$. When the time t is clear, we also use $\hat{R}$ to denote $\hat{R}^{(t)}$.

a) Updating Information in the Decomposition Tree.: We next describe the algorithm to update all information pertaining to $\mathcal{T}$ that we need to detect which parts of $\mathcal{T}$ to rebuild and that are useful when constructing the induced emulator. We give our algorithm in Algorithm 3.

Algorithm 3: UpdateInformation(t)

```
1: Let $e = (x,y)$ be the edge updated at time $t$ in $G$.
2: for $(R,S) \in \mathcal{T}$ where $e \in E(\hat{R})$ and $e \notin E(\hat{R}_i)$ for all $(R_i, S_i) \in \mathtt{Ch}(R,S)$ do
3:     Portals$_{(R,S)} \leftarrow$ Portals$_{(R,S)} \cup \{x,y\}$.
4: end for
5: for $(R,S) \in \mathcal{T}$ of Type-2-3 where $e$ is a valid edge update to $\hat{R}^{(t-1)}$ do
6:     Let $\pi$ be the shortest path in $R$ that was computed when the children of $(R,S)$ were instantiated in Line 16 or Line 29 of Algorithm 2. Portals$_{(R,S)} \leftarrow$ Portals$_{(R,S)} \cup \Big($SourcePathPortals$(R, \pi, \{x,y\}, \epsilon') \cap \hat{R}\Big)$.
7: end for
```

The algorithm updates the portal sets of affected instances. In particular, there are two types of being affected for an instance: if (R,S) (or rather $\hat{R}^{(t-1)}$) undergoes an edge insertion but none of its children do, then (R,S) is responsible for the new edge e in our algorithm. We therefore need to add the endpoints x,y of the inserted edge e to the portal set of (R,S). The second Type-of being affected is when (R,S) is of Type-2 or 3 and the updated edge is a valid edge update to $\hat{R}^{(t-1)}$. While in this case, child instances still might contain the edge, we need to compute additional portals on the shortest-path separator that we used to decompose (R,S) further, which ensures that portals close to the endpoints of e are added to these separators.

D. Updating the Decomposition Tree

Finally, we show how to use the updated information to adjust the decomposition tree $\mathcal{T}$. We have already discussed that for instance $(R, S) \in \mathcal{T}$ that we denote by $\hat{R}$ the graph obtained from simulating relevant updates to G on R (see Definition V.1). In this section, we also have to look at how the set of terminals S evolves. We denote by $\hat{S}$ the dynamic set of vertices that are terminals for $\hat{R}$. Thus, $(\hat{R}, \hat{S})$ denotes the instance that is evolving from $(R, S) \in \mathcal{T}$.

Let us now define $\hat{S}$ formally. This is rather straightforward: we want the original vertices of S in $\hat{S}$ as long as they still appear in $\hat{R}$. And additionally, if a strict ancestor (R', S') of (R, S) in the decomposition tree $\mathcal{T}$ places a new portal into $\textsc{Portals}_{(R',S')}$ and this portal appears in $\hat{R}$, then it should be added to the terminal set $\hat{S}$. We give the following formal definition.

Definition V.2. *At any time, for any instance $(R, S) \in \mathcal{T}$, we define*

$$\hat{S} = (S \cap V(\hat{R})) \cup \left(\bigcup_{(R',S') \text{ is a strict ancestor of } (R,S)} \textsc{Portals}_{(R',S')} \cap V(\hat{R}) \right)$$

The goal of our procedure to update the decomposition tree is to ensure that for any instance (R, S) in $\mathcal{T}$, we have that R and $\hat{R}$ do not differ in too many edges, and that $\hat{S} \setminus S$ remains much smaller than S. This will ensure that our recursion depth remains bounded by $O(\log n)$ which is crucial for our algorithm. If either of these assumptions is violated, we remove the subtree of $\mathcal{T}$ rooted at (R, S), denoted by $\mathcal{T}[(R, S)]$, and then add $(\hat{R}, \hat{S})$ as a child to (R, S)'s former parent and run the initialization procedure (see Algorithm 1) on this instance.

For convenience, we also define sets $\textsc{AncPortals}_{(R,S)}$ and $\textsc{StrictAncPortals}_{(R,S)}$ where the latter one is just $\hat{S} \setminus S$, while the former one also contains the portal set of (R, S) itself. For technical reasons, we have to work with the former set, however, in the algorithm's analysis, we use both of these definitions heavily.

Definition V.3. *At any time, for any instance $(R, S) \in \mathcal{T}$, we define*

$$\textsc{AncPortals}_{(R,S)} := (\hat{S} \cup (\textsc{Portals}_{(R,S)} \cap V(\hat{R}))) \setminus S$$
$$\textsc{StrictAncPortals}_{(R,S)} := \hat{S} \setminus S.$$

It is easy to make the following observation which we exploit later in the analysis.

Observation V.4. *At any time, for any instance $(R, S) \in \mathcal{T}$ with parent (R', S'), we have $(\textsc{AncPortals}_{(R',S')} \cap V(\hat{R})) \setminus S = \textsc{StrictAncPortals}_{(R,S)}$.*

Algorithm 4: $\textsc{UpdateDecompositionTree}(t)$

```
for $(R, S) \in \mathcal{T}$ where $|E(\hat{R}) \Delta E(R)| \geq |V(R)| \cdot \sqrt{\frac{\tau}{n}}$ do
    Replace the sub-tree $\mathcal{T}[(R, S)]$ by $(\hat{R}, \hat{S})$ (if it is the root, then it stays the root).
    $\textsc{InitializeInstance}((\hat{R}, \hat{S}), t)$.
end for
for $(R, S) \in \mathcal{T}$ where $(R, S)$ is of Type-2-3 and $|\textsc{AncPortals}_{(R,S)}| + |\mathcal{P}^{\text{trivial}}_{(R,S)}| > \frac{1}{12 \log^2 n}|S|$ or $(R, S)$ is of Type-4 and $|\textsc{AncPortals}_{(R,S)}| + |\mathcal{P}^{\text{trivial}}_{(R,S)}| > \sqrt{\frac{\tau}{n}} \cdot |V(R)|$ do
    Replace the sub-tree $\mathcal{T}[(R, S)]$ by $(\hat{R}, \hat{S})$ (if it is the root, then it stays the root).
    $\textsc{InitializeInstance}((\hat{R}, \hat{S}), t)$.
end for
```

Finally, we can give the pseudo-code for our update procedure of the decomposition tree (see Algorithm 4). Again, it either replaces an instance (R, S) by $(\hat{R}, \hat{S})$ due to $\hat{R}$ diverging too much from R (see the foreach-statement starting in Line 1), or if the set of ancestor portals $\textsc{AncPortals}_{(R,S)}$ grows too large which intuitively implies that $\hat{S}$ diverges heavily from S (see the foreach-statement starting in Line 1).

E. Initializing the Induced Planar Emulator

Recall that the main goal of our algorithm is to maintain a set of paths $\mathcal{P}$ that are non-crossing in a graph $\tilde{G}$ obtained from recursively performing slicing operations in G (see Fact III.2). We first describe how we define the set of paths $\mathcal{P}$. We then define the graph $\tilde{G}$ which naturally arises in the process.

In our algorithm, we obtain the collection of paths $\mathcal{P}$ bottom-up from the decomposition tree $\mathcal{T}$. We therefore associate with each instance $(R, S) \in \mathcal{T}$ an associated set of paths $\mathcal{P}_{(R,S)}$. We then define $\mathcal{P} = \mathcal{P}_{(G,\emptyset)}$.

We define these collections as follows.

a) Type-1 Path Collection.: For every instance $(R, S) \in \mathcal{T}$ of Type-1, we let $\mathcal{P}_{(R,S)}$ be the set of non-crossing paths obtained via the algorithm from Theorem III.5 w.r.t $\hat{S}$ on graph $\hat{R}$.

b) Type-2 Path Collection.: For every instance $(R, S) \in \mathcal{T}$ of Type-2, let $(R_1, S_1), (R_2, S_2), \ldots, (R_k, S_k)$ be the children of (R, S) in $\mathcal{T}$. As we prove in the full version, we maintain that $\hat{R}$ is a one-hole instance and that $f_\infty(\hat{R}) \supseteq f_\infty(R)$, i.e. the hole of the evolved graph $\hat{R}$ contains the hole of the original graph R of the instance. We further have for children $(\hat{R}_i, \hat{S}_i)$ that

- $\hat{R}_i \subseteq \hat{R}$,
- $(\hat{R}_i, \hat{S}_i)$ is a one-hole instance and in every graph $\hat{R}_j$ for $j \neq i$, we have that the interior of $\hat{R}_j$ is contained in the hole $f_\infty(\hat{R}_i)$ (here the interior of $\hat{R}_j$ is defined to be the interior of the complement of the closed region $f_\infty(\hat{R}_j)$),
- all paths in $\mathcal{P}_{(R_i,S_i)}$ are in the complement of $f_\infty(\hat{R}_i)$ (Induction and Theorem III.5),
- $\mathcal{M}_G[\mathcal{P}_{(R_i,S_i)}]$ preserves the distances between vertices in $\hat{S}_i$ approximately.

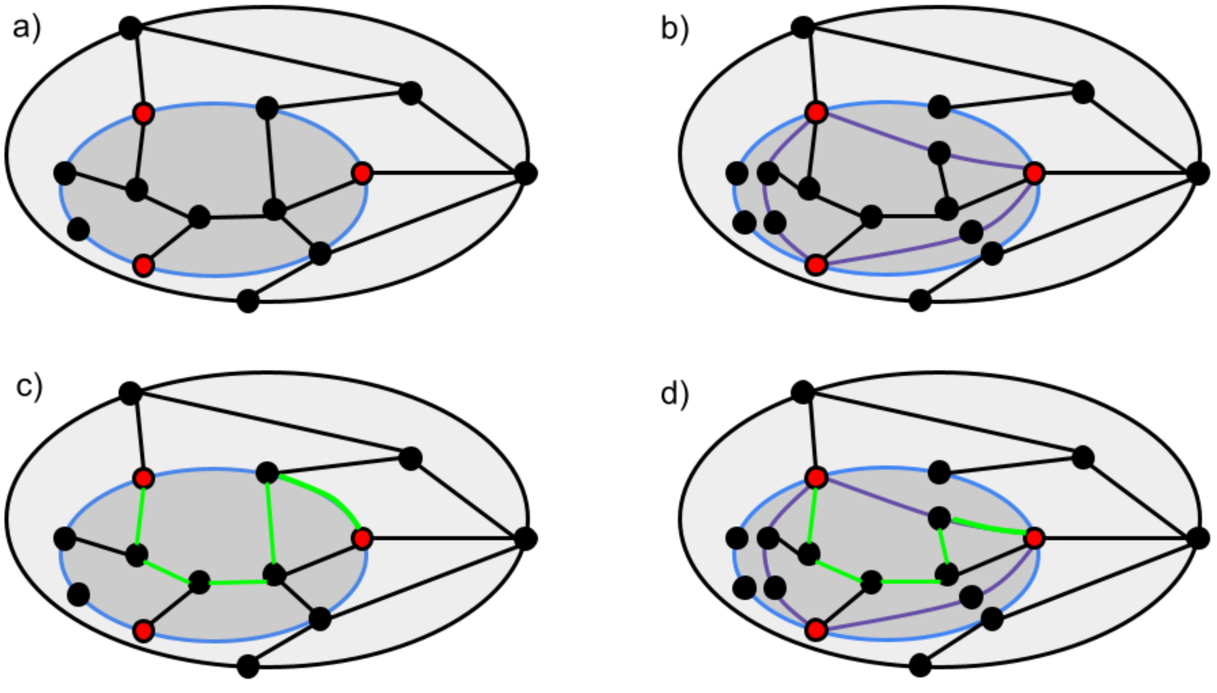


Fig. 7: Illustration of Slicing Procedure of a Type-2 instance (R, S) with child (R_i, S_i). All figures show in light grey the graph $\hat{R}$ and in dark grey the graph $\hat{R}_i[C]$ for some connected component C in $\hat{R}_i$. Vertices of $\hat{S}_i$ are red, and all other vertices are black. In a) we show the graph $\hat{R}$ before the slicing procedure, we have the cycle enclosing the hole of $\hat{R}_i[C]$ in blue. In b) we show the graph obtained from performing the slicing procedure for $\hat{R}_i[C]$ and have in lilac all edges that were added during the slicing procedure. In c) we give a path P from $\mathcal{P}_{(R_i,S_i)}$ that is contained in $\hat{R}_i[C]$. In d), we show how the path P is re-mapped into the graph after the slicing procedure and we have that this new path is added to $\mathcal{P}_{(R,S)}$.

Then, we obtain $\mathcal{P}_{(R,S)}$ by the following procedure (see also Figure 7): for every $1 \leq i \leq k$, and connected component C of $\hat{R}_i$ where $C \cap \hat{S}_i$ is of size at least two, we partition the cycle that encloses the hole $f_\infty(\hat{R}_i[C])$ into maximal paths $P_1, P_2, \ldots, P_k$ that contain no internal vertex in $\hat{S}_i$ (note that we prove that all vertices in $\hat{S}_i \cap C$ are on $f_\infty(\hat{R}_i[C])$ in the full version). Then, we slice along these paths $P_1, P_2, \ldots, P_k$, such that the newly added paths are all added to be in the interior $\hat{R}_i$. Finally, for every path $P \in \mathcal{P}_{(R_i,S_i)}$ that is contained in $\hat{R}_i[C]$, we add to $\mathcal{P}_{(R,S)}$ the path obtained from mapping P into the new graph where every edge on P that appears on some path P_j is mapped to the corresponding copy on the path P'_j that is in the interior of $\hat{R}_i[C]$. We point out that all slicing procedures can be performed independently, i.e. they do not affect each other and the outcome is not altered by applying them in different order.

Finally, we add for each edge $e \in E(\hat{R}) \setminus (E(\hat{R}_1) \cup E(\hat{R}_2) \cup \ldots E(\hat{R}_k))$, a trivial path consisting just of the edge e to $\mathcal{P}_{(R,S)}$. We denote these trivial paths as $\mathcal{P}^{\text{trivial}}_{(R,S)}$.

Note that the above slicing procedure ensures that the paths in $\mathcal{P}_{(R,S)}$ are non-crossing. That is because each $\hat{R}_i[C]$ with at least two vertices in $\hat{S}_i$ on $f_\infty(\hat{R}_i)$ has that $\mathcal{P}_{(R_i,S_i)}$ contains for every vertex $s \in \hat{S}_i$ at least one path that starts or ends in s, otherwise distances between these vertices could not be approximately preserved since $\hat{R}_i[C]$ is connected by definition. But note that by the definition of non-crossing shortest paths, we have that this implies that no vertex in $\hat{S}_i \cap C$ being internal to any path in $\mathcal{P}_{(R_i,S_i)}$. It remains to observe that the process of re-mapping paths from $\mathcal{P}_{(R_i,S_i)}$ into the graph obtained from slicing retains the property that these paths are pairwise non-crossing in $\mathcal{P}_{(R,S)}$. And no vertex on the hole $f_\infty(\hat{R}_i[C])$ for any connected component C in $\hat{R}_i$ contains any vertex that is internal to any path in $\mathcal{P}_{(R,S)}$ that is obtained from re-mapping a path from $\mathcal{P}_{(R_i,S_i)}$. Thus, any two paths in $\mathcal{P}_{(R,S)}$ that are from different graphs $\hat{R}_i[C]$ and $\hat{R}_j[C']$ where i, j not necessarily distinct if C and C' aren't, intersect at most in their endpoints. Thus, $\mathcal{P}_{(R,S)}$ is again a collection of non-crossing paths.

c) Type-3 Path Collection.: For every instance $(R, S) \in \mathcal{T}$ of Type-3, let $(R_1, S_1), \ldots, (R_k, S_k)$ be the children of (R, S) in $\mathcal{T}$.

Recall that when (R, S) is initialized, due to being of Type-3, it adds to $\mathcal{T}$ only a single child (R', S') that is obtained from slicing R along a shortest path π_{uv} between two holes of R (see the statement starting in Line 27 of Algorithm 1 and also Figure 1 for an illustration).

Since the update algorithm (see Algorithm 3) only replaces children (R', S') by themselves before calling the procedure

INITIALIZEINSTANCE$(\cdot)$ (see Algorithm 1), we have that for all children of (R, S), we have that the graphs of its children instances, i.e. $R_1, R_2, \ldots, R_k$, are edge-disjoint except for the edges on the shortest path that was used to slice R to obtain R' which appear at most twice across all such graphs. And since $S_1, S_2, \ldots, S_k$ are supersets of the set S induced on $R_1, R_2, \ldots, R_k$ respectively, we have for every instance (R_i, S_i), and every connected component C in $\hat{R}_i$ with at least two vertices in $\hat{S}_i \cap C$, that no paths P in collection $\mathcal{P}_{(R_i,S_i)}$ that is in $\hat{R}[C]$ has any of the vertices in $\hat{S} \cup \text{PORTALS}_{(R,S)}$ as an internal vertex.

Thus, we take $\mathcal{P}_{(R,S)}$ to consist of all paths from any collection $\mathcal{P}_{(R_i,S_i)}$ that is contained in a connected component with at least two vertices in $\hat{S}$. And finally, we add for each edge $e \in E(\hat{R}) \setminus (E(\hat{R}_1) \cup E(\hat{R}_2) \cup \ldots E(\hat{R}_k))$, a trivial path consisting just of the edge e to $\mathcal{P}_{(R,S)}$. We denote these trivial paths as $\mathcal{P}^{\text{trivial}}_{(R,S)}$.

By our former reasoning, we have that $\mathcal{P}_{(R,S)}$ is a collection of non-crossing paths.

d) Type-4 Path Collection.: Finally, for any instance (R, S) of Type-4 with children $(R_1, S_1), (R_2, S_2), \ldots, (R_k, S_k)$, we take $\mathcal{P}_{(R,S)}$ to consist of all paths in $\mathcal{P}_{(R_i,S_i)}$ for all i where $\hat{S}_i$ is of size at least 2, and again the collection of trivial paths that contains for every $e \in E(\hat{R}) \setminus (E(\hat{R}_1) \cup E(\hat{R}_2) \cup \ldots E(\hat{R}_k))$, a trivial path consisting just of the edge e to $\mathcal{P}_{(R,S)}$. We denote these trivial paths as $\mathcal{P}^{\text{trivial}}_{(R,S)}$.

This is again a collection of non-crossing paths as the graphs $\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_k$ and the collection of edges not in any of these paths are all edge-disjoint. Further, all boundary vertices between these graphs are shown to be in $\hat{S}_i$ for every i, and thus all paths in $\mathcal{P}_{(R_i,S_i)}$ cannot have any vertex in $\hat{S} \cup \text{PORTALS}_{(R,S)}$ as an internal vertex.

e) The graph $\tilde{G}$.: Finally, we also need to maintain the graph $\tilde{G}$ that contains all paths in our final path collection $\mathcal{P}_{(G,\emptyset)}$. But note that we can simply take $\tilde{G}$ as the union of all edges on any such path. Here, the role of $\tilde{G}$ is to clarify the embedding of the edges on paths, however, the embedding is already clear from the above definitions.

F. Maintaining Induced Planar Emulators Efficiently

Both initialization and maintenance of our planar emulator induced by collection $\mathcal{P}_{(G,\emptyset)}$ is rather straightforward: we run a bottom-up algorithm over the nodes of $\mathcal{T}$ and construct each path collection as described above. After every update, we identify the nodes in $\mathcal{T}$ that were changed at the current time or where information was changed at the current time. We then re-compute the path collection in a bottom-up fashion over the affected nodes and their parents.

As we show in the analysis, the above algorithm is efficient as every update can only change a few instances, and the information maintained of few instances in every tree. Further, Type-1, Type-2, and Type-3 nodes have instance graphs of small size, and therefore re-computation is not too expensive. For Type-4 nodes, we essentially take the union of path collections of children which can be done efficiently as only a few of these rather small collections change overall.

G. Adding the Terminal Set T to the Emulator

Note that the collection of paths $\mathcal{P}_{(G,\emptyset)}$ technically does not preserve distances between any pair of vertices. However, after initialization and processing every update, we run an additional step that adds the vertices in T to the emulator to ensure that the distances between vertices in T are preserved.

Therefore, we invoke Algorithm 5. This algorithm updates the information at nodes in $\mathcal{T}$ similarly to how information is updated by Algorithm 3. Our algorithm adds every terminal $t \in T$ to every portal set of an instance graph that contains t. Finally, we update the induced planar emulator and the path collection $\mathcal{P}_{(G,\emptyset)}$ based on this new information (note that we do not rebuild any parts of $\mathcal{T}$).

Algorithm 5: ADDTERMINALS()

```
1: for $t \in T$ do
2:     for $(R, S) \in \mathcal{T}$ where $t \in V(\hat{R})$ but $t \notin S$ do
3:         $\text{PORTALS}_{(R,S)} \leftarrow \text{PORTALS}_{(R,S)} \cup \{t\}$.
4:     end for
5:     for $(R, S) \in \mathcal{T}$ of Type-2-3 where $t \in V(\hat{R})$ do
6:         Let $\pi$ be the shortest path in $R$ computed during Line 16 or Line 29 of Algorithm 2.
7:         $\text{PORTALS}_{(R,S)} \leftarrow \text{PORTALS}_{(R,S)} \cup \left(\text{SOURCEPATHPORTALS}(R, \pi, \{t\}, \epsilon') \cap \hat{R}\right)$.
8:     end for
9: end for
```

Finally, when the next update to G arrives, we first revert the effect of the last invocation of ADDTERMINALS() also on the emulator. Then, we process the update as discussed above and finally again invoke ADDTERMINALS() to update the emulator to preserve distances between terminals in T. Clearly, the run-time and recourse of adding a terminal are identical to that of the information update algorithm.

REFERENCES

[1] Faster algorithms for finding small edge cuts in planar graphs (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 229–240. ACM, 1992.

[2] Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023.

[3] Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in p via short cycle removal: cycle detection, distance oracles, and beyond. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1487–1500. ACM, 2022.

[4] Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486, 2016.

[5] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

[6] Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 740–753. SIAM, 2016.

[7] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1199–1218. ACM, 2012.

[8] Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the o(n) barrier. In *APPROX-RANDOM*, volume 28 of *LIPIcs*, pages 1–16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[9] Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *FOCS*, pages 693–702. IEEE Computer Society, 2009.

[10] Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.*, 45(2):548–574, 2016.

[11] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. *arXiv preprint arXiv:2101.07149*, 2021.

[12] Parinya Chalermsook, Jittat Fakcharoenphol, and Danupon Nanongkai. A deterministic near-linear time algorithm for finding minimum cuts in planar graphs. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 828–829. SIAM, 2004.

[13] Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenkovic, Shay Solomon, and Cuong Than. Resolving the steiner point removal problem in planar graphs via shortcut partitions. *CoRR*, abs/2306.06235, 2023.

[14] Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Almost-linear ε-emulators for planar graphs. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1311–1324, 2022.

[15] Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Near-linear ϵ-emulators for planar graphs. *arXiv preprint arXiv:2206.10681*, 2022.

[16] Hsien-Chih Chang and Tim Ophelders. Planar emulators for monge matrices. In J. Mark Keil and Debajyoti Mondal, editors, *Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada*, pages 141–147, 2020.

[17] Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 138–151, 2019.

[18] Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately - lower and upper bounds. In *ICALP*, volume 55 of *LIPIcs*, pages 131:1–131:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[19] Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1159–1172. ACM, 2023.

[20] Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973. IEEE, 2017.

[21] Ingemar J. Cox, Satish Rao, and Yu Zhong. "ratio regions": a technique for image segmentation. In *13th International Conference on Pattern Recognition, ICPR 1996, Vienna, Austria, 25-19 August, 1996*, pages 557–564. IEEE Computer Society, 1996.

[22] Debarati Das, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. A near-optimal offline algorithm for dynamic all-pairs shortest paths in planar digraphs. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3482–3495. SIAM, 2022.

[23] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato Fonseca F. Werneck. Customizable route planning. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.

[24] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina Anna Zweig, editors, *Algorithmics of Large and Complex Networks - Design, Analysis, and Simulation [DFG priority program 1126]*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

[25] Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In Camil Demetrescu, editor, *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, volume 4525 of *Lecture Notes in Computer Science*, pages 52–65. Springer, 2007.

[26] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

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

[28] Arnold Filtser. Steiner point removal with distortion o(log k) using the relaxed-voronoi algorithm. *SIAM J. Comput.*, 48(2):249–278, 2019.

[29] Arnold Filtser. Scattering and sparse partitions, and their applications. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 47:1–47:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[30] Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1226–1245. SIAM, 2021.

[31] Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 50:1–50:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[32] Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen. Truly subquadratic exact distance oracles with constant query time for planar graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[33] Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 61:1–61:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[34] Davi Geiger, Alok Gupta, Luiz A. Costa, and John A. Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(3):294–302.

[35] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[36] Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^{ϵ} worst-case update time. *CoRR*, abs/2402.18541, 2024.

[37] Refael Hassin and Donald B. Johnson. An o(n $\log^2$ n) algorithm for maximum flow in undirected planar networks. *SIAM J. Comput.*, 14(3):612–624, 1985.

[38] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016.

[39] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.

[40] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *journal of computer and system sciences*, 55(1):3–23, 1997.

[41] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[42] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[43] Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM Journal on Computing*, 8(2):135–150, 1979.

[44] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322. ACM, 2011.

[45] Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017.

[46] Adam Karczmarz. Decrementai transitive closure and shortest paths for planar digraphs and beyond. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 73–92. SIAM, 2018.

[47] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 81–91. IEEE Computer Society, 1999.

[48] Philip Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 820–827, 2002.

[49] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155. SIAM, 2005.

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

[51] Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

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

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

[54] Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications. *CoRR*, abs/2311.06402, 2023.

[55] Jakub Lacki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in o(n loglogn) time. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2011.

[56] Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 363–374. IEEE, 2022.

[57] Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2517–2537. SIAM, 2021.

[58] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995.

[59] Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012.

[60] Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In Camil Demetrescu, editor, *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2007.

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

[62] Jan van den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1011–1022. IEEE, 2022.