Single-Source Shortest Paths and Strong Connectivity in Dynamic Planar Graphs

Panagiotis Charalampopoulos
Department of Informatics, King's College London, UK
Institute of Informatics, University of Warsaw, Poland
panagiotis.charalampopoulos@kcl.ac.uk

Adam Karczmarz
Institute of Informatics, University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Abstract

Efficient algorithms for computing and processing additively weighted Voronoi diagrams on planar graphs have been instrumental in obtaining several recent breakthrough results, most notably the almost-optimal exact distance oracle for planar graphs [Charalampopoulos et al., STOC'19], and subquadratic algorithms for planar diameter [Cabello, SODA'17, Gawrychowski et al., SODA'18]. In this paper, we show how Voronoi diagrams can be useful in obtaining dynamic planar graph algorithms and apply them to classical problems such as dynamic single-source shortest paths and dynamic strongly connected components.

First, we give a fully dynamic single-source shortest paths data structure for planar weighted digraphs with $\widetilde{O}(n^{4/5})$ worst-case update time and $O(\log^2 n)$ query time. Here, a single update can either change the graph by inserting or deleting an edge, or reset the source s of interest. All known non-trivial planarity-exploiting *exact* dynamic single-source shortest paths algorithms to date had polynomial query time. Further, note that a data structure with strongly sublinear update time capable of answering distance queries between all pairs of vertices in polylogarithmic time would refute the APSP conjecture [Abboud and Dahlgaard, FOCS'16].

Somewhat surprisingly, the Voronoi diagram based approach we take for single-source shortest paths can also be used in the fully dynamic strongly connected components problem. In particular, we obtain a data structure maintaining a planar digraph under edge insertions and deletions, capable of returning the identifier of the strongly connected component of any query vertex. The worst-case update and query time bounds are the same as for our single-source distance oracle. To the best of our knowledge, this is the first fully dynamic strong-connectivity algorithm achieving both sublinear update time and polylogarithmic query time for an important class of digraphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shortest paths

Keywords and phrases dynamic graph algorithms, planar graphs, single-source shortest paths, strong connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.31

Funding *Panagiotis Charalampopoulos*: Partially supported by ERC Starting Grant TOTAL under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 677651).
Adam Karczmarz: Supported by ERC Consolidator Grant 772346 TUgbOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 31; pp. 31:1–31:23

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The dynamic shortest paths problem seeks for a data structure maintaining a graph under updates and supporting shortest path queries.[1] Depending on the set of supported updates, we call such a graph data structure *fully dynamic* if both edge insertions and deletions are allowed, *incremental* if only edge insertions (or edge weight decreases) are supported, or *decremental* if only edge deletions (or weight increases) are allowed. In the *all-pairs* variant of dynamic shortest paths problem one has to support shortest path queries between any pair of vertices of the graph. In the *single-source* variant all shortest paths queries have to originate in a fixed distinguished vertex and the only parameter of a query is the target vertex.

For the most general setting where one requires *exact* answers, Demetrescu and Italiano [31] gave a fully dynamic algorithm (improved slightly by Thorup [77]) recomputing the all-pairs shortest paths matrix in nearly optimal $\widetilde{O}(n^2)$ amortized time even if real edge weights are allowed. Note that recomputing the distance matrix from scratch takes $\widetilde{O}(nm) = \widetilde{O}(n^3)$ time [55]. Fully dynamic all-pairs shortest paths data structures with subcubic *worst-case* update bounds are also known [5, 46, 78]. There exist faster algorithms if the input graph is unweighted and *partially dynamic* (i.e., incremental or decremental) [6, 7]. However, none of the known results improves upon a trivial, recompute-from-scratch algorithm with $\widetilde{O}(mn)$ update time and $O(1)$ query time if the graph is *sparse*, i.e., $m = \widetilde{O}(n)$. For the single-source variant, all known non-trivial exact dynamic shortest paths algorithms [37] are partially dynamic and yield no improvement over the respective recompute-from-scratch algorithm in the sparse case either.

The lack of progress on obtaining an exact fully dynamic single-source shortest paths algorithms with $O(n^{3-\epsilon})$ initialization time, $O(m^{1-\epsilon})$ amortized update time and $O(n^{1-\epsilon})$ query time at the same time can be explained by a matching lower bound conditional on the (static) APSP conjecture [73]. In fact, breaking this barrier even in the partially dynamic setting for undirected unweighted graphs would be a large breakthrough [43].

As a result, since finding good exact algorithms for general graphs seems hopeless, one needs to look for either approximate solutions or restrict their attention to more structured graph classes. Indeed, a large body of research has been devoted to designing *approximate* dynamic shortest paths algorithms, especially in partially dynamic settings [9–13, 15, 26, 29, 43–45, 48–50, 59, 60], which find many applications, e.g., in various maximum flow related problems [29, 67]. Unfortunately, many of the known fastest approximate dynamic shortest path algorithms (e.g. [9,44,48]) suffer from assuming an oblivious adversary, which significantly limits their applicability (cf. e.g., [29]).

Similarly, dynamic shortest paths problems have also been studied for important graph classes like planar graphs [3, 4, 38, 56, 63, 66], or low treewidth-graphs [3, 58]. The primary reason why faster dynamic shortest paths algorithms in these cases are possible is the existence of non-trivial *distance oracles* for these classes. A distance oracle is a compact representation of the graph's shortest paths such that the distance (or a distance estimate) between any pair of vertices can be retrieved efficiently. For general graphs, such non-trivial distance oracles exist only for undirected graphs and assuming an approximation ratio of at least 3 [25, 79, 82]. On the contrary, for planar graphs many non-trivial exact distance oracles

[1] We will identify shortest paths queries with distance queries. Almost all known dynamic shortest paths algorithms (for some exceptions see [74, 80]) can also report the actual path in nearly linear (in the number of the path's edges) time after computing a distance estimate.

have been proposed [19,27,33,38,56,63,69]. The first exact oracles with *polylogarithmic* query time and subquadratic space have been obtained only recently [23,30,41], following Cabello's breakthrough of employing Voronoi diagrams for the planar diameter problem [20]. Also near-optimal (in terms of query time, construction time, and used space) $(1+\epsilon)$-approximate distance oracles have been known for nearly two decades [62,76], and a lot of effort has been put to push the known bounds as close to optimal as possible [22,42,61,84].

Dynamic shortest paths in planar graphs. In this paper our focus is on computing shortest paths in dynamic planar graphs and applications. Klein and Subramamian were the first to give a planarity-exploiting dynamic shortest paths algorithm [66] – their data structure worked for undirected graphs, was fully dynamic, $(1+\epsilon)$-approximate and had $\widetilde{O}(n^{2/3})$ update and query time bounds. A data structure with the same bounds (up to polylogarithmic factors), but for *exact* distances in *directed* graphs was obtained in the breakthrough work of Fakcharoenphol and Rao [38] (later extended and slightly improved in [24,40,54,56,63]). Abraham et al. [4] gave a faster $(1+\epsilon)$-approximate dynamic algorithm for undirected graphs with $\widetilde{O}(n^{1/2})$ update and query times. Karczmarz [58] matched this bound for directed planar graphs, albeit only in the $(1+\epsilon)$-approximate decremental setting. Abboud and Dahlgaard [1] showed that by the APSP conjecture, one should not expect an exact dynamic all-pairs shortest paths data structure for planar graphs with strongly sublinear product of update time and query time. However, no exact data structure to date has matched this product lower bound while retaining strongly sublinear update time.

The single-source scenario is much less studied for dynamic planar graphs. Karczmarz [58] showed a decremental $(1+\epsilon)$-approximate single-source shortest paths algorithm for minor-free (and thus also planar) digraphs with $\widetilde{O}(n^{1/2})$ update time and $O(1)$ query time. Although not explicitly stated in the literature, the all-pairs data structure of [63] can be easily converted to a fully dynamic exact single-source distance oracle with $\widetilde{O}(n^{2/3})$ update time and $\widetilde{O}(n^{1/3})$ query time. However, no *fully dynamic* single-source shortest paths algorithm for planar graphs to date has been able to achieve sublinear update time and polylogarithmic query time, or at least break through the $\widetilde{O}(n)$ update-query time product barrier, even in the approximate setting.

Our results. In this paper we show the first exact dynamic single-source shortest paths algorithm for planar graphs with *strongly sublinear* update time and *polylogarithmic* query time. Our algorithm, summarized by the following theorem and described in Section 3, is deterministic and can be easily extended to report paths.

▶ **Theorem 1.** *Let G be a real-weighted planar digraph with a source $s \in V(G)$. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions, edge deletions, and source changes with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute $dist_G(s,v)$ for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

To the best of our knowledge, this result constitutes the first known application of additively weighted Voronoi diagrams machinery (first introduced by Cabello [20]) in dynamic graph algorithms. More specifically, it is obtained by combining fully dynamic maintenance of r-divisions [66,75], the shortest paths algorithm for dense distance graphs [38], the recent efficient construction of dual Voronoi diagrams [23] via FR-Dijkstra [38], and the efficient point location data structure for Voronoi diagrams [41].

We now provide a brief overview of the data structure underlying Theorem 1. We maintain distances in G from the source vertex s to each boundary vertex of each piece of an r-division of G using FR-Dijkstra. For each piece of the r-division, we maintain an additively weighted

Voronoi diagram augmented with a point location data structure, with weights equal to the distances from s. Upon a query for $\text{dist}_G(s, v)$, we perform a point location query on the Voronoi diagram of a piece of the r-division that contains v.

It is worth noting that our data structure (and in fact all data structures obtained in this paper) works in the most general model of dynamic planar graphs where we only require that G remains planar after each update. Some fully dynamic planar graph algorithms assume a weaker *plane* model (e.g., [32, 35, 54]) where some plane embedding of G is fixed and we only allow inserting edges connecting vertices that lie on a common face of (that embedding of) G.

We also generalize our single-source data structure to the case when, instead of a single source s, a set of *facilities* $F \subseteq V$ is given, and our goal is to locate the closest (i.e., minimizing $\text{dist}_G(f, v)$) facility $f \in F$ for a given query vertex.[2] We show that maintaining such a data structure under edge updates issued to G, or updates to the facilities set F, is possible using $\widetilde{O}(n^{3/4} \cdot |F|^{1/4} + n^{4/5})$ worst-case update time. The query time remains $O(\log^2 n)$. Note that even though multiple-source shortest paths or maximum flow problems can be typically easily reduced to the single-source case by adding a super-source, such a reduction does not preserve planarity and indeed handling multiple sources tends to be challenging in planar graphs (cf. e.g., [16, 17]). Our generalized data structure handles up to $O(n^{1/5})$ sources as efficiently as the single-source case. Moreover, the update time remains strongly sublinear unless the number of facilities is not strongly sublinear.

Surprisingly, we show that the same framework that we use to prove Theorem 1 can be applied to obtain interesting results not directly related to the shortest paths problem. Namely, in Section 4 we show a fully dynamic strong-connectivity algorithm for planar graphs, encapsulated in the following theorem.

▶ **Theorem 2.** *Let G be a planar digraph. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions and deletions with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute the identifier of the strongly connected component of any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

We now sketch the main ideas behind our fully dynamic strong-connectivity algorithm. As in Subramanian's dynamic all-pairs reachability algorithm [75], the base of our data structure is a graph X, called a *reachability certificate*, that sparsifies the reachability information between boundary vertices $\partial\mathcal{R}$ of a fully dynamic r-division $\mathcal{R}$ with few holes. Naively recomputing the strongly connected components of X gives us the restriction of the strongly connected components of G to the boundary vertices $\partial\mathcal{R}$. The main challenge, of course, is to compute the identifier of a strongly connected component (SCC) of an arbitrary non-boundary vertex v of G, internal to some piece P of the r-division $\mathcal{R}$. To this end, we use the following observation: suppose $b_1, \ldots, b_k$ are some vertices of G lying in distinct strongly connected components of G. Then, v is strongly connected to some b_j if and only if b_j is in the topologically earliest SCC of G reachable from v *and* b_j is in the topologically latest SCC of G that can reach v. Roughly speaking, this observation applied to the boundary vertices of P labeled using the topological order of their respective SCCs in the certificate X, allows us to identify the SCC of v using two point-location queries on the Voronoi diagram of piece P. Each such point location query, computes, instead of the nearest site of v, the highest (or lowest) priority site that can reach v (that v can reach, resp.), and can be simulated using a standard point location query on a Voronoi diagram [41].

[2] One can also view F as a set of sites of a graphic Voronoi diagram – then the query locates the cell of the Voronoi diagram wrt. F that a given vertex v belongs to.

Whereas maintaining strongly connected components is a well-studied problem in partially dynamic settings [8, 14, 47, 53], we are not aware of any non-trivial fully dynamic strongly connected components data structures designed specifically for this problem for *any* digraph class – note that one could use a fully dynamic transitive closure data structure for this task: for example, the dynamic plane transitive closure data structure of [32] which has $\widetilde{O}(n^{1/2})$ update and query time. Such a strongly connected components data structure (i.e., with both update and query bounds $O(n^{1-\epsilon})$) for *general graphs* is in fact ruled out by a conditional (on SETH) lower bound [2]. As a result, to the best of our knowledge, we obtain the first fully dynamic strongly connected components algorithm to achieve sublinear update-query time product for *any* important class of digraphs.

The undirected counterpart of the dynamic strongly connected components problem, the *dynamic connectivity* problem, is very well-studied. Near-optimal deterministic amortized update bounds [51,52,83] and randomized worst-case update bounds [57,81] (see also [71]) are known for fully dynamic general graphs. An almost optimal deterministic worst-case update bound was very recently achieved in [28]. For fully dynamic planar graphs polylogarithmic worst-case update bounds are known to be achievable even deterministically [34].

2 Preliminaries

Throughout the paper we consider as input a simple, directed and weighted planar graph G with n vertices, and no negative weight cycles. We call a planar graph G *plane* if some embedding of G is assumed. We use $|G|$ to denote the number of vertices of G. Since simple planar graphs are sparse, $|E(G)| = O(|G|)$ as well.

We use the terms weight and length for edges and paths interchangeably throughout the paper. For any two vertices $u, v \in V(G)$, we denote by $\mathrm{dist}_G(u, v)$ the length of some shortest $u \to v$ path in the graph G.

Multiple-source shortest paths. The multiple-source shortest paths (MSSP) data structure [21, 63] represents all shortest path trees rooted at the vertices of a single face f in a weighted plane digraph using a persistent dynamic tree. It can be constructed in $O(n \log n)$ time, requires $O(n \log n)$ space, and can report any distance between a vertex of f and any other vertex in the graph in $O(\log n)$ time. MSSP can be augmented to also return the first edge of this path (and each of its subsequent edges) in $O(\log \log n)$ time (cf. [56]).

Separators and recursive decompositions. Miller [68] showed how to compute, in a triangulated plane graph with n vertices, a simple cycle of size $2\sqrt{2}\sqrt{n}$ that separates the graph into two subgraphs, each with at most $2n/3$ vertices. Simple cycle separators can be used to recursively separate a planar graph until pieces have constant size. The authors of [64] show how to obtain a complete recursive decomposition tree $\mathcal{T}(G)$ of a triangulated graph G using cycle separators in $O(n)$ time. $\mathcal{T}(G)$ is a binary tree whose nodes correspond to subgraphs of G (pieces), with the root being all of G and the leaves being pieces of constant size. We identify each piece P with the node representing it in $\mathcal{T}(G)$. We can thus abuse notation and write $P \in \mathcal{T}(G)$. The *boundary vertices* ∂P of a *non-leaf piece* P are vertices that P shares with some other piece $Q \in \mathcal{T}(G)$ that is not P's ancestor. For convenience we extend the boundary set ∂L of a leaf piece L to its entire vertex set $V(L)$. We assume P to inherit the embedding of G. The faces of P that are faces of G are called *natural*, whereas the faces of P that are not natural are the *holes* of P. The construction of [64] additionally guarantees that for each piece $H \in \mathcal{T}(G)$, (a) H is connected, (b) if H is non-leaf, then each natural

face f of H is a face of a unique child of H, (c) H has $O(1)$ holes containing precisely the vertices ∂H. Throughout, to avoid confusion, we use *nodes* when referring to $\mathcal{T}(G)$ and *vertices* when referring to G or its subgraphs. It is well-known [18, 41, 53, 64] that by suitably choosing cycle separators one can also guarantee that (1) $\sum_{H\in\mathcal{T}(G)} |H| = O(n \log n)$, (2) $\sum_{H\in\mathcal{T}(G)} |\partial H|^2 = O(n \log n)$, and (3) $|\partial H| = O(\sqrt{n}/c^d)$, where node H of $\mathcal{T}(G)$ has depth d and $c > 1$ is some constant.

The recursive decomposition algorithm of [64] works with no changes and maintains all the properties of $\mathcal{T}(G)$ even if the initial graph G has a predefined set of boundary vertices ∂G of size $O(\sqrt{|G|})$ located on $O(1)$ of G's faces. These faces are predefined as holes of G and are the only faces of G that are allowed to be non-triangular.

An *r-division* [39] $\mathcal{R}$ of a planar graph, for $r \in [1, n]$, is a decomposition of the graph into $O(n/r)$ pieces, each of size $O(r)$, such that each piece P has $O(\sqrt{r})$ boundary vertices (denoted ∂P), i.e., vertices shared with some other piece of $\mathcal{R}$. We denote by $\partial\mathcal{R}$ the set $\bigcup_{P\in\mathcal{R}} \partial P$. If additionally all pieces are connected, and the boundary vertices of each piece P of the r-division $\mathcal{R}$ are distributed among $O(1)$ faces of P (also called holes[3] of P), we call $\mathcal{R}$ an *r-division with few holes*.

In [64] it was shown that for every r larger than some constant, $\mathcal{T}(G)$ *admits* an r-division with few holes, i.e., there exists a subset of nodes of $\mathcal{T}(G)$ forming an r-division with few holes of G. Using this property, it is shown in [64] that an r-division with few holes of a triangulated graph can be computed in linear time. More generally, given a geometrically decreasing sequence of numbers $(r_m, r_{m-1}, \ldots, r_1)$, where r_1 is a sufficiently large constant, $r_{i+1}/r_i \geq b$ for all i for some $b > 1$, and $r_m = n$, we can obtain r_i-divisions with few holes for all i in time $O(n)$ in total. For convenience, we define the only piece in the r_m-division to be G itself. These r-divisions satisfy the property that a piece in the r_i-division is a – not necessarily strict – descendant (in $\mathcal{T}(G)$) of a piece in the r_j-division for each $j > i$. We also call such sequence of r_i-divisions obtained from $\mathcal{T}(G)$ a recursive $(r_m, \ldots, r_1)$-division of G.

We assume for simplicity that all holes we ever encounter are simple cycles. Unfortunately, this is not true in general. However, non-simple holes do not pose a significant obstacle, and can be avoided by suitably extending the graphs, as discussed numerous times in the past, see e.g., [23, 53, 56, 72].

Dense distance graphs and FR-Dijkstra. For a plane digraph H with weights from $\mathbb{R}_{\geq 0} \cup \{\infty\}$ and a distinguished set $\partial H \subseteq V(H)$ of boundary vertices lying on $O(1)$ faces of H, we denote by DDG_H the complete weighted graph on ∂H whose edge weights represent distances between all pairs of vertices of ∂H in H. DDG_H can be computed in $O\left((|H| + |\partial H|^2) \log n\right)$ time using MSSP [63]. In particular, dense distance graphs for all pieces $H \in \mathcal{T}(G)$ can be computed in $O(n \log^2 n)$ time.

When $\mathcal{H} = \{H_1, \ldots, H_q\}$ is a collection of plane graphs, we set $\mathrm{DDG}(\mathcal{H}) := \bigcup_{H\in\mathcal{H}} \mathrm{DDG}_H$.

▶ **Lemma 3** (FR-Dijkstra [38, 40]). *Given all DDG_{H_i}, one can compute a single-source shortest paths tree from any source s in $\mathrm{DDG}(\mathcal{H})$ in $O\left(\sum_{i=1}^{q} |\partial H_i| \log^2 n\right)$ time, where $n = |V(\mathrm{DDG}(\mathcal{H}))|$.*[4]

[3] This definition is slightly more general than the definition of a hole of a piece $P \in \mathcal{T}(G)$. Namely, the definition of an r-division does not assume a fixed embedding of the entire G; it only assumes some fixed embeddings of individual pieces.

[4] In particular H_i may be single-edge. This way, this lemma captures also the case when we compute shortest paths in a collection of DDGs with some auxiliary vertices and edges.

We now state some fairly standard definitions and lemmas about representing distances between some vertices of G of interest using unions of dense distance graphs of a recursive decomposition's (or r-division's) pieces (for instance cf. [24]), adapted to our notation. We include their proofs for completeness in Appendix A. For example, Lemma 4 captures the well-known observation of [38] that in order to compute a shortest path between any pair of vertices of G, it is enough to compute a shortest path in a union of dense distance graphs from $\mathcal{T}(G)$ with only $O(\sqrt{n})$ vertices in total.

Let L be some leaf of $\mathcal{T}(G)$. We define the *cone* of L, denoted $\mathrm{cone}_G(L)$, to be the collection of pieces of $\mathcal{T}(G)$ containing L, all ancestors of L, and all siblings of (weak) ancestors of L. For some collection $\mathcal{L}$ of leaf pieces of $\mathcal{T}(G)$, we define $\mathrm{cone}_G(\mathcal{L}) = \bigcup_{L\in\mathcal{L}} \mathrm{cone}_G(L)$.

▶ **Lemma 4** ([24, 38]). *Let $\mathcal{L}$ be some collection of leaf pieces of $\mathcal{T}(G)$. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L})))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L}))}(u,v)$.*
2. $\sum_{H\in\mathrm{cone}_G(\mathcal{L})} |\partial H| = O\left(\sqrt{n|\mathcal{L}|}\right)$.

Let $\mathcal{R}$ be an r-division with few holes of G and $\mathcal{T}(P)$ be a recursive decomposition of $P \in \mathcal{R}$ with the root boundary set to ∂P. For any $v \in V(G) \setminus \partial G$, let L_v be some leaf containing v in the unique piece $P_v \in \mathcal{R}$ containing v. For any $X \subseteq V(G)$ let us define $\mathrm{cone}_{\mathcal{R}}(X) = \mathcal{R} \cup \bigcup_{v\in X\setminus\partial\mathcal{R}} \mathrm{cone}_{P_v}(L_v)$.

▶ **Lemma 5** ([24, 38]). *Let $X \subseteq V(G)$ be non-empty. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X)))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X))}(u,v)$.*
2. $\sum_{H\in\mathrm{cone}_{\mathcal{R}}(X)} |\partial H| = O\left(n/\sqrt{r} + \min\left(\sqrt{n\cdot|X|}, |X|\cdot\sqrt{r}\right)\right)$.

Fully dynamic r-divisions. Many dynamic algorithms for planar graphs maintain r-divisions and useful auxiliary data structures under dynamic updates. The exact set of supported updates to G varies; e.g., [38,56,63] support only edge weight changes, [54] assumes embedding-preserving insertions, whereas [66, 75] only assume that the graph G remains planar at all times. We stick to the last, most general setting. The core of the construction behind the following theorem is due to Klein and Subramanian [66, 75]; for completeness we give a complete proof in Appendix A.

▶ **Theorem 6.** *Let $G = (V, E)$ be a weighted planar graph. Suppose that adding infinite-weight edges to G does not have effect on any properties of G that we care about. Let $r \in [1, n]$.*

There is a data structure maintaining an r-division with few holes $\mathcal{R}$ of some G^+ such that:

1. *G^+ is obtained from G by adding infinite-weight edges.*
2. *Each P has all its faces except its holes triangular and is accompanied with some auxiliary data structures that can be constructed in $T(r)$ time given P and use $S(r)$ space.*

The data structure uses $O\left(n + \frac{n}{r}\cdot S(r)\right)$ space and can be initialized in $O\left(n + \frac{n}{r}\cdot T(r)\right)$ time. After each edge deletion and edge insertion (preserving the planarity of G), it can be updated in $O(r + T(r))$ worst-case *time.*

Additively weighted Voronoi diagrams. Let G be a directed planar graph of size n with real edge-lengths, and no negative-length cycles. Assume that all faces of G are triangles except, perhaps, a single face f. Let S be the set of vertices that lie on f, called *sites*, i.e., $S = V(f)$. Let us assign to each site $s \in S$ a weight $\omega(s) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$. The additively weighted distance $\mathrm{dist}^{\omega}_G(s, v)$ between a site $s \in S$ and a vertex $v \in V(G)$ is defined as $\omega(s) + \mathrm{dist}_G(s, v)$.

The additively weighted Voronoi diagram of (S, ω) within G, denoted by $\mathsf{VD}(S, \omega)$, is a partition of $V(G)$ into pairwise disjoint sets, one set $\mathsf{Vor}(s)$ for each site $s \in S$. The set $\mathsf{Vor}(s)$, which is called the Voronoi cell of s, contains all vertices in $V(G)$ that are closer (wrt. $\mathrm{dist}^{\omega}_{G}$) to s than to any other site in S.

In the following and throughout, whenever we work with Voronoi diagrams we assume that (1) G is strongly connected, (2) shortest paths in G are unique, and (3) additively weighted shortest paths in G are unique, i.e., for each $v \in V(G)$ there is a unique site s minimizing $\mathrm{dist}^{\omega}_{G}(s, v)$. Note that these assumptions make the Voronoi cells well-defined and simply connected, and guarantee that they indeed form a partition of $V(G)$. We will explicitly ensure that these requirements are met for G, S and the weight function whenever we define a Voronoi diagram on G.

There is a dual representation $\mathsf{VD}^*(S, \omega)$ of Voronoi diagram $\mathsf{VD}(S, \omega)$ as a tree of constant degree with $O(|S|)$ vertices and edges [41]. An efficient FR-Dijkstra based algorithm for computing $\mathsf{VD}^*(S, \omega)$ was presented by Charalampopoulos et al. [23].

▶ **Theorem 7** ([23]). *Suppose that we have at hand a recursive decomposition $\mathcal{T}(G)$ of G, with the only hole of G being f and $S = V(f)$. Further suppose that we have DDG_H computed for each piece $H \in \mathcal{T}(G)$. Then, we can compute $\mathsf{VD}^*(S, \omega)$ in $O(\sqrt{n \cdot |S|} \log^2 n)$ time.*

▶ Remark 8. The algorithm underlying Theorem 7 implicitly assumes that $\mathsf{Vor}(s)$ is non-empty for all $s \in S$. In Appendix B, we discuss why this assumption is not necessary, relying on [41].

In a *point location* query for some Voronoi diagram $\mathsf{VD}(S, \omega)$, we are given a vertex $v \in V(G)$ and are requested to find the site $s \in S$ such that $v \in \mathsf{Vor}(s)$ and also the value of $\mathrm{dist}^{\omega}_{G}(s, v)$. Gawrychowski et al. [41] showed the following result.

▶ **Theorem 9** ([41]). *Suppose that we have at hand an MSSP data structure for G with sources from the face f. Given some dual representation $\mathsf{VD}^*(S, \omega)$, we can preprocess it in $O(|S|)$ time, so that point location queries for $\mathsf{VD}(S, \omega)$ can be answered in $O(\log^2 n)$ time.*

3 Fully Dynamic Single Source Shortest Paths

In this section we show our single-source exact distance oracle for planar graphs with $O(n^{4/5} \log^2 n)$ update time and $O(\log^2 n)$ query time and thus prove Theorem 1. For simplicity, let us assume that G is non-negatively weighted. Negative edges can be handled as in [56] – see Appendix C.

▶ **Theorem 1.** *Let G be a real-weighted planar digraph with a source $s \in V(G)$. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions, edge deletions, and source changes with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute $\mathrm{dist}_G(s, v)$ for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

The base of our data structure is a dynamic r-division $\mathcal{R}$ with few holes, as given in Theorem 6. Note that in our shortest-paths problem, indeed adding infinite-weight edges to G does no harm. Hence, in the following we work with the graph G^+ from Theorem 6 when computing distances, but identify it, without loss of generality, with our original graph G.

For technical reasons, however, we would like to avoid dealing with infinite weights in some of our data structures handling individual pieces. In the real-weighted fully dynamic setting, however, we cannot fix a sufficiently large finite number, larger than all edge weights that will ever appear in the future graph G, beforehand. Instead, we do the following. For

each $P \in \mathcal{R}$, let M_P be a sufficiently large finite number, e.g., larger than the sum of finite edge weights in P. Consider M_P to be an auxiliary data structure of P as in Theorem 6. We will use M_P to simulate infinite edge weights in P, and also for detecting paths non-existent in the original graph G (but having infinite weight in $G^+ \cap P$). As a result, below we assume each infinite weight in P (or any auxiliary data structure related to P) is replaced by M_P in all the computations performed locally on the piece P, whereas globally (when performing some computation for many pieces at once, like the shortest paths algorithm of Lemma 3) we treat all edge or path weights in P that are at least M_P as infinite.

For each piece $P \in \mathcal{R}$ we store the following additional data structures.

- We store the recursive decomposition $\mathcal{T}(P)$ with the initial boundary set to ∂P, and also DDGs for all the pieces $H \in \mathcal{T}(P)$.
- For each hole h of P, let P_h be the piece P after applying the following standard augmentations. First, P is extended into a graph P'_h using $O(r)$ vertices and edges of weight M_P embedded inside either the piece or other (than h) holes of P that would make P strongly connected and triangulated (except for the hole h) without changing the distance between any pair of reachable vertices in P. The graph P_h is in turn obtained from P'_h by changing P'_h's edge weights into $O(1)$-size vectors as described in [36] so that there is a *unique* shortest path (wrt. the lexicographical order on path weights, defined as the coordinate-wise sum of the path's individual edge weights) between any $u, v \in V(P_h)$ with cost of the form $(\mathrm{dist}_{P'_h}(u, v), \cdot)$. As proven in [36], one can compute P_h from P deterministically in linear time. The $O(1)$-size vector weights, in turn, can be easily packed into usual single-number weights.

  For each P_h, we store an MSSP data structure initialized for the hole h. Recall that an MSSP data structure can be computed in $O(r \log r)$ time. Moreover, we store a recursive decomposition $\mathcal{T}(P_h)$ of P_h with the boundary ∂P_h of the root piece set to $\partial P \cap V(h)$ of size $O(\sqrt{r})$. For each node (piece) $H \in \mathcal{T}(P_h)$, we also store DDG_H. Since the sum of sizes of all the pieces of $\mathcal{T}(P_h)$ is $O(r \log r)$, computing all these dense distance graphs takes $O(r \log^2 r)$ time (see Section 2).

Note that computing piecewise auxiliary data structures defined so far takes $O(r \log^2 r)$ time. So, by Theorem 6, they can be updated in $O(r \log^2 r)$ worst-case time after G undergoes an update.

After the initialization and each update, once $\mathcal{R}$ and all auxiliary data structures are updated, we compute for each $P \in \mathcal{R}$ a point location data structure.

▶ **Lemma 10.** *Given a weight function $\omega : \partial P \to \mathbb{R} \cup \infty$, one can compute in $O(r^{3/4} \log^2 r)$ time a data structure $L(P)$ answering the following queries in $O(\log^2 r)$ time: given any $v \in V(P)$, compute the value $\min_{b \in \partial P}\{\omega(b) + dist_P(b, v)\}$ along with the minimizer b.*

Proof. Since ∂P is a union of $O(1)$ sets ∂P_h, we can compute the desired minimum over each ∂P_h separately and then take the minimum over all h.

Let us first note that negative values of ω are not a problem. We can turn negative weights into non-negative by adding some common large value to the weights of all sites. We can thus suppose wlog. that all values of ω are non-negative.

If all the weights are infinite, the queries can be answered trivially in $O(1)$ time. So in the following assume that there is at least one site whose weight is finite.

Let $S = \partial P_h = \{s_1, \ldots, s_k\}$ be the set of sites. In order to guarantee that for each $v \in V(G)$ there is a unique site s minimizing $\mathrm{dist}^{\omega}_G(s, v)$, we will break ties by considering $(\omega(s_i), i)$ instead of $\omega(s_i)$ as the weight of site s_i, adding a second coordinate to each edge weight in P_h, set to 0, and comparing additively weighted distances lexicographically. Clearly, this extension does not break any of the properties of P_h.

Recall that P_h has finite non-negative real weights, is strongly connected, has unique shortest paths, and has a single face h that is possibly non-triangular that contains all the sites S. Moreover, the additively weighted distances in P_h are unique. Therefore, we can invoke Theorem 7 to construct the dual representation $\mathsf{VD}^*(S,\omega)$ of the Voronoi diagram $\mathsf{VD}(S,\omega)$. This requires $O\left(\sqrt{r|\partial P_h|}\log^2 n\right) = O\left(r^{3/4}\log^2 n\right)$ time.

Then, we construct the point location data structure of Theorem 9 for $\mathsf{VD}(S,\omega)$. Note that we have an MSSP data structure for P_h for hole h and hence point location queries, given $\mathsf{VD}^*(S,\omega)$, can be answered in $O(\log^2 n)$ time. ◀

We invoke Lemma 10 with weight function $\omega(b) := \mathrm{dist}_G(a,b)$ in order to construct $L(P)$ for each $P \in \mathcal{R}$. This requires $O\left(n/r^{1/4}\cdot\log^2 n\right)$ time in total. By Lemma 5, the values $\mathrm{dist}_G(s,b)$ for all $b \in \partial R$ can be computed in $O\left(n/\sqrt{r}\cdot\log^2 n\right)$ time if we run the single-source shortest paths algorithm of Lemma 3 (FR-Dijkstra) on the graph $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$. We also compute $\mathrm{dist}_{P_s}(s,u)$ for all $u \in P_s$, where P_s is an arbitrary piece containing s using Dijkstra's algorithm in $O(r\log r)$ time.

Now, we can compute $\mathrm{dist}_G(s,v)$ for a query vertex v as follows. If the shortest $s \to v$ path in G does not go through $\partial\mathcal{R}$, then it is fully contained in P_s and therefore $v \in P_s$ and $\mathrm{dist}_G(s,v) = \mathrm{dist}_{P_s}(s,v)$, i.e., we have $\mathrm{dist}_G(s,v)$ already computed. Otherwise, let P_v be an arbitrary piece containing v. Observe that we have $\mathrm{dist}_G(s,v) = \min_{b\in\partial P_v}\{\mathrm{dist}_G(s,b) + \mathrm{dist}_{P_v}(b,v)\}$ where the minimizer b corresponds to the a boundary vertex of some shortest $s \to v$ path in G. So this case can be reduced to a single query to the data structure $L(P_v)$. This takes $O(\log^2 n)$ time.

The worst-case update time is $O(r\log^2 r + \frac{n}{r^{1/4}}\log^2 r)$. By setting $r = n^{4/5}$ we get $O(n^{4/5}\log^2 n)$ worst-case update time. Since the space usage per piece is $O(r\log r)$, we need $O(n\log n)$ space.

▶ Remark 11. Our data structure can be extended to report, following the computation of $\mathrm{dist}_G(s,v)$, a shortest $s \to v$ path Q in time nearly linear in the number of edges of Q. This follows easily by the fact that the MSSP data structure [63] can report shortest paths efficiently (see e.g., [56] for details). Therefore, we can efficiently expand the used edges of dense distance graphs and the shortest $b \to v$ path into actual edges in G.

3.1 A Dynamic Closest Facility Data Structure

We can generalize the dynamic single-source shortest paths data structure as follows. Suppose we replace a single source vertex s with a set of *facilities* $F \subseteq V$. Given F, for a query vertex v we would like to compute $\min_{f\in F}\{\mathrm{dist}_G(f,v)\}$, and also possibly $f \in F$ minimizing this expression. A dynamic update would consist of either an edge update or changing the set F. In other words, such a problem can be seen as dynamic point location in a Voronoi diagram wrt. F, where each update either changes the graph or resets the Voronoi diagram of interest.

In this setting, we consider the following simple generalization of the single-source data structure. Let the update procedure first compute the distances $d(b) = \min_{f\in F}\{\mathrm{dist}_G(f,b)\}$ for all $b \in \partial\mathcal{R}$. Note that by Lemmas 5 this can be achieved by computing single-source shortest paths in the graph $\mathrm{DDG}(\mathcal{D}_F)$, where $\mathcal{D}_F = \mathrm{cone}_{\mathcal{R}}(F)$, extended with 0-weight edges sf, where s is an auxiliary super-source. By Lemma 3 this can be done in $O((\sqrt{n|F|}+n/\sqrt{r}+|F|)\log^2 n) = O((\sqrt{n|F|}+n^{4/5})\log^2 n)$ time. By using weights $\omega := d$ in the individual point-location data structures $L(P)$, $P \in \mathcal{R}$, a single point location query on $L(P_v)$ (recall that P_v is some piece containing v) would compute the desired closest facility f_v minimizing $\mathrm{dist}_G(f_v,v)$ unless the sought (weighted) shortest $f_v \to v$ does not go through a boundary

vertex of $\mathcal{R}$. We could in principle handle such paths by proceeding as in the single-source case and computing shortest paths naively in each piece containing a facility. However, this could take time $\Omega(r \cdot \min(n/r, |F|))$, i.e., linear in n even for moderately large facility set sizes, e.g., $|F| = \Omega(n^{1/5})$.

To improve upon this simple approach, we proceed as follows. Let $(\rho_m, \ldots, \rho_1)$ be such a sequence of integers that $\rho_m = r$, $\rho_1 = O(1)$ and $\rho_{i+1}/\rho_i = 2$ for all $i < m$. For each $P \in \mathcal{R}$ we store a recursive $(\rho_m, \ldots, \rho_1)$-division consisting of pieces of $\mathcal{T}(P)$ (cf. Section 2). Let $\mathcal{R}_{P,i}$ be the ρ_i-division of P. All $\mathcal{R}_{P,i}$ can be computed in linear time given $\mathcal{T}(P)$ [64]. Note that $\mathcal{R}_i$, defined as the union of $\mathcal{R}_{P,i}$ over all pieces $P \in \mathcal{R}$, actually forms an ρ_i-division with few holes of the entire graph G. In particular, we have $\mathcal{R}_m = \mathcal{R}$.

We store the extended pieces Q_h (recall how we obtained extended pieces P_h with unique shortest paths from P in the single-source case) plus their recursive decompositions $\mathcal{T}(Q_h)$, DDGs, and an MSSP data structure for all pieces Q of *all* $\mathcal{T}(P)$ instead of just the pieces of $\mathcal{R}_m = \mathcal{R}$ as we did in the single-source case. However, we stress that these auxiliary components for a piece $Q \subseteq P$ where $P \in \mathcal{R}$, are counted as accompanying data structures of the piece P. So, we compute $O(1)$ fresh recursive decompositions $\mathcal{T}(Q_h)$ for each piece $Q \in \mathcal{T}(P)$ – computing each takes $O(|Q| \log^2 n)$ time. As a result, by the bound $\sum_{Q \in \mathcal{T}(P)} |Q| = O(|P| \log |P|)$, the time to compute accompanying data structures of piece P increases to $O(r \log^3 n)$.

Given the set of facilities F, let j be such that $\rho_j = \Theta\left(\min\left(\frac{n}{|F|}, r\right)\right)$. Redefine $\mathcal{D}_F = \mathrm{cone}_{\mathcal{R}_j}(F)$. Again, let us compute distances $d(b) = \min_{f \in F}\{\mathrm{dist}_G(f, b)\}$ (and the closest facilities) for all $b \in \bigcup_{H \in \mathcal{D}_F} |\partial H|$ using FR-Dijkstra on DDG$(\mathcal{D}_F)$ extended with a super-source s and auxiliary edges sf, $f \in F$. This takes $O\left(\left(\sqrt{n|F|} + n/\sqrt{\rho_j}\right) \log^2 n\right) = O(\sqrt{n|F|} \log^2 n)$ time by Lemmas 3 and 5.

It only remains to show how to handle computation of closest facilities for $v \in V \setminus \bigcup_{H \in \mathcal{D}_F} \partial H$. Recall that the pieces $\mathcal{D}_F$ cover the entire G, no facility f is an internal (non-boundary) vertex of a piece $H \in \mathcal{D}_F$, and each $v \in V \setminus \bigcup_{H \in \mathcal{D}_F} \partial H$ is clearly an internal vertex of a unique piece $H_v \in \mathcal{D}_F$. Consequently, by Lemma 10, the closest facility to v can be found in $O(\log^2 n)$ time using a single query to the data structure $L(H_v)$. For this to be possible, upon update we need to build the data structures $L(H)$ for all $H \in \mathcal{D}_F$. By Lemma 10, this takes $O\left(\sum_{H \in \mathcal{D}_F} \sqrt{|H| \cdot |\partial H|} \log^2 n\right)$ time. Let us now bound this sum. First, let us consider the sum restricted to the pieces $H \in \mathcal{D}_F \cap \mathcal{R}_j$, i.e., the pieces of r-division $\mathcal{R}_j$. Since $\rho_j = \Omega(n/|F|)$ or $\rho_j = \Omega(r)$ we get:

$$O\left(\sum_{H \in \mathcal{D}_F \cap \mathcal{R}_j} \sqrt{|H| \cdot |\partial H|} \log^2 n\right) = O\left(\frac{n}{\rho_j} \cdot \rho_j^{3/4} \log^2 n\right) = O\left(\left(n^{3/4} \cdot |F|^{1/4} + \frac{n}{r^{1/4}}\right) \log^2 n\right).$$

On the other hand, if $H \notin \mathcal{D}_F \cap \mathcal{R}_j$, then $H \in \mathrm{cone}_{P_f}(L_f)$, where $f \in F \setminus \mathcal{R}_j$, P_f is the unique piece of $\mathcal{R}_j$ containing f, and L_f is some leaf of $\mathcal{T}(P_f)$ containing f. For a fixed f, by the definition of $\mathrm{cone}_{P_f}(L_f)$, there are $O(\log n)$ pieces H satisfying this, at most two per each level i of $\mathcal{T}(P_f)$. Hence, the sum of $\sqrt{|H| \cdot |\partial H|}$ over such pieces can be bounded by $\sum_{i=0}^{\infty} \sqrt{|P_f| \cdot \sqrt{|P_f|}/c^i} = O(\rho_j^{3/4})$. Summing over all f, and using $\rho_j = O(n/|F|)$, we get

$$O\left(\sum_{H \in \mathcal{D}_F \setminus \mathcal{R}_j} \sqrt{|H| \cdot |\partial H|} \log^2 n\right) = O\left(|F| \cdot \rho_j^{3/4} \log^2 n\right) = O\left(n^{3/4} \cdot |F|^{1/4} \log^2 n\right).$$

To conclude, the update time is $O\left(\left(n^{3/4} \cdot |F|^{1/4} + \frac{n}{r^{1/4}}\right) \log^2 n + r \log^3 n\right)$. By setting $r = (n/\log n)^{4/5}$ we obtain the following theorem.

▶ **Theorem 12.** *Let G be a real-weighted planar digraph with a set $F \subseteq V$ of facilities. There exists an $O(n \log^2 n)$-space data structure maintaining G under edge insertions, edge deletions, and changes of the facilities set F with $O\left(\left(n^{3/4} \cdot |F|^{1/4} + n^{4/5}\right) \log^{11/5} n\right)$ worst-case update time that can compute $\min_{f \in F}\{dist_G(f, v)\}$ along with the respective closest facility of v for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^3 n)$.*

4 Fully Dynamic Strongly Connected Components

In this section we show that a strategy similar to that of Section 3 can be used to obtain a fully dynamic strong-connectivity algorithm. Again, we maintain an r-division $\mathcal{R}$ and some auxiliary data structures for all the individual pieces. Formally, using a dynamic r-division as in Theorem 6 may require introducing new infinite-weight edges to G which in turn may change the reachability relation in G. We circumvent this problem by setting the weights of the original edges of G to 0, and all auxiliary edges plus infinity (simulated in the implementation by sufficiently large values M_P inside individual pieces, as in Section 3). This way, u can reach v in G if and only if $\text{dist}_G(u, v) = 0$, and otherwise $\text{dist}_G(u, v) = \infty$. All known properties of reachability in plane graphs also extend to reachability using 0-weight paths (assuming non-negative weights). In the following, whenever we say that v is reachable from u, or there exists a $u \to v$ path, we mean $\text{dist}_G(u, v) = 0$.

As in Section 3, for each piece of $\mathcal{R}$ we store a recursive decomposition, dense distance graphs and MSSP data structures. All these data structures are also maintained for pieces of $\mathcal{R}$ with all edges reversed – for a piece P we call this graph the *reverse* of P and denote it by P^{rev}.

Another ingredient is a collection of *reachability certificates* X_P for all the pieces, as defined in the following lemma due to Subramanian [75], slightly adjusted to certify 0-weight paths.

▶ **Lemma 13** ([75])**.** *Let $P \in \mathcal{R}$ be a piece. There exists a directed graph X_P, where $\partial P \subseteq V(X_P)$, of size $O(\sqrt{r} \log r)$ satisfying the following property: for any $u, v \in \partial P$, $dist_P(u, v) = 0$ if and only if there exists a $u \to v$ path in X_P. The graph X_P can be computed in $O(r \log r)$ time.*

We include the reachability certificate in the set of auxiliary piecewise data structures. Since reachability certificates can be computed in $O(r \log r)$ time, maintaining them does not incur any additional asymptotic cost. The following lemma is a direct consequence of Lemma 13.

▶ **Lemma 14.** *For any $u, v \in \partial\mathcal{R}$, u can reach v in G if and only if u can reach v in $X = \bigcup_{P \in \mathcal{R}} X_P$.*

Proof. Let $u, v \in \partial\mathcal{R}$. Since each X_P certifies the reachability between ∂P in P, clearly a $u \to v$ path in X implies an existence of a $u \to v$ path in G. Now suppose there is a $u \to v$ path Q in G. Split P into maximal subpaths $Q_1, \ldots, Q_k$, such that each Q_i is fully contained in a single piece $P_i \in \mathcal{R}$. For each i, the endpoints a, b of Q_i are contained in ∂P_i and hence there exists a $a \to b$ path in $X_P \subseteq X$. Consequently, there exist a $u \to v$ path in X. ◀

To handle an edge update, after $\mathcal{R}$ and auxiliary data structures are updated, we compute the strongly connected components of X (defined as in Lemma 14) in $O(|X|) = O(n/\sqrt{r})$ time using any classical linear-time algorithm. For any $b \in \partial\mathcal{R}$, let $s_X(b)$ denote an integer

identifier of b's strongly connected component in X. By additionally sorting the SCCs of X topologically we can further assume that s_X satisfies the following property: if $a, b \in \partial\mathcal{R}$ are not strongly connected, but a can reach b in X then $s_X(a) < s_X(b)$. By Lemma 14, for $a, b \in \partial\mathcal{R}$, we have $s_X(a) = s_X(b)$ if and only if a and b are strongly connected in G; moreover, if a can reach b in G, then $s_X(a) \leq s_X(b)$.

We also define and maintain similar SCC-identifiers s_P for individual pieces P, i.e., for $u, v \in V(P)$, $s_P(u) = s_P(v)$ implies u, v are strongly connected in P, whereas $s_P(u) < s_P(v)$ implies there is no $v \to u$ path in P. Clearly, the identifiers s_P can be recomputed in $O(r)$ time given P, so we also include them into the set of auxiliary per-piece data structures.

For any $Q \in \{X\} \cup \mathcal{R}$, let S_Q be the set of used identifiers of the form $s_Q(\cdot)$. Observe that we can easily guarantee that the sets S_Q are pairwise disjoint, e.g., by using disjoint integer ranges for different sets S_Q.

The final component of our data structure, is, again a collection of per-piece point location data structures. For each $P \in \mathcal{R}$, we have two point location data structures $L(P^{\text{rev}})$ and $L(P)$ of Lemma 10. After each edge update, $L(P^{\text{rev}})$ is computed for P^{rev} with weight function $\omega = s_X$. On the other hand, $L(P)$ is initialized with weight function $\omega = -s_X$. As in Section 3, all these point location data structures are recomputed in $O(n/r^{1/4} \cdot \log^2 n)$ time (over all pieces).

We now describe how our data structure handles a query for an SCC identifier of a vertex v. The returned identifier always comes from the set $\bigcup_{Q \in \{X\} \cup \mathcal{R}} S_Q$. Let P_v be some piece containing v. Let $s_{\min}$ be the value computed by $L(P_v^{\text{rev}})$ for vertex v. Let $s_{\max}$ be *minus* the value computed by $L(P_v)$ for v. If either of $s_{\min}, s_{\max}$ equals $\pm\infty$ or $s_{\min} \neq s_{\max}$ holds, we return $s_{P_v}(v)$. Otherwise, we return $s_{\min} \in S_X$. The following lemma establishes the correctness of this query procedure.

▶ **Lemma 15.** *Let $u, v \in V(G)$ and let s_u, s_v be the respective identifiers returned by the query procedure. Then, $s_u = s_v$ if and only if u and v are strongly connected in G.*

Proof. Suppose $s_u = s_v$. If $s_u \in S_P$ for some piece P, then $s_u = s_v$ implies that u and v are strongly connected in P and thus also in G. So suppose $s_v \in S_X$. Take any $b \in \partial\mathcal{R}$ such that $s_v = s_X(b)$. We now prove that v and b are strongly connected in G. Similarly we prove that u and b are strongly connected in G. By transitivity it will follow that u and v are indeed strongly connected.

Let $P_v, s_{\min}, s_{\max}$ be defined as in the query procedure's description. Recall that $s_v \in S_X$ implies that $s_{\min}, s_{\max}$ are finite and $s_v = s_{\min} = s_{\max}$. Since all edges of P_v have weight 0, and $s_{\max}$ is finite, $s_{\max}$ in fact represents the maximum value $s_X(a)$ among those $a \in \partial P_v$ such that a path $a \to v$ exists in P_v. Similarly, observe that $s_{\min}$ represents the minimum value $s_X(c)$ among those $c \in \partial P_v$ such that a path $c \to v$ exists in P_v^{rev}, i.e., such that a path $v \to c$ exists in P_v. Let us denote by a and c the respective vertices of ∂P_v attaining the maximum and minimum values of s_X. Since $s_X(a) = s_X(c)$, there exists a path $c \to a$ in G. However, by the definition of a and c, paths $a \to v$ and $v \to c$ also exist in G, and hence a, v and c are strongly connected in G. Since a is clearly strongly connected to b by $s_X(a) = s_X(b)$, indeed v and b are strongly connected in G.

Now let us move to proving the " $\Longleftarrow$ " direction. Suppose u and v are strongly connected in G. First consider the case when there exists some vertex $b \in \partial\mathcal{R}$ located in the same strongly connected component of G as u and v. In this case we prove that $s_v = s_X(b)$. An analogous proof that $s_u = s_X(b)$ will establish $s_u = s_v$. Since v and b are strongly connected, there exist some paths $Q_1 = v \to b$ and $Q_2 = b \to v$ in G. Let v_1 be the first vertex on Q_1 such that $v_1 \in \partial P_v$ – note that v_1 necessarily exists since $b \in \partial\mathcal{R}$. Similarly set v_2 to be

the last vertex on Q_2 such that $v_2 \in \partial P_v$. Observe that the subpaths $v \to v_1$ and $v_2 \to v$ of Q_1 and Q_2 respectively lie entirely inside P_v. Hence, $s_{\min}$ and $s_{\max}$ are finite and we have $s_{\max} \geq s_X(v_2)$ and $s_{\min} \leq s_X(v_1)$. Recall that there exists a walk $v \to v_1 \to b \to v_2 \to v$, so in fact v_1, v_2, b are strongly connected, i.e., $s_X(v_1) = s_X(v_2) = s_X(b)$. Thus, we obtain $s_{\min} \leq s_X(b) \leq s_{\max}$.

On the other hand, let $a \in \partial P_v$ be such that a path $a \to v$ exists in P_v and $s_{\max} = s_X(a)$ (a exists by $s_{\max} \neq \pm\infty$). Similarly, let $c \in \partial P_v$ be such that a path $v \to c$ exists in P_v and $s_{\min} = s_X(c)$. Since a path $a \to c$ through v exists in P_v (so also in G), we have that $s_{\max} \leq s_{\min}$ by the fact that the identifiers S_X respect the topological order of the SCCs of X. Recall that we have already proved $s_{\min} \leq s_X(b) \leq s_{\max}$ so in fact we have $s_{\min} = s_{\max} = s_X(b)$, and consequently $s_v = s_X(b)$.

Finally, suppose there is no vertex of $\partial\mathcal{R}$ in the SCC of G containing u and v. First, this implies that $u, v \notin \partial\mathcal{R}$ and all $u \to v$ and $v \to u$ paths are contained in a single, unique piece P. This implies that $s_P(u) = s_P(v)$. Hence it is sufficient to prove $s_u = s_P(u)$ and $s_v = s_P(v)$. We prove the latter equality; proving the former is analogous. Recall that s_v is *not* set to $s_P(v)$ only if both $s_{\min}, s_{\max}$ are finite and $s_{\min} = s_{\max}$. This can only happen if there exists vertices $a, c \in \partial P$ such that a can reach v in P, v can reach c in P and $s_X(c) = s_{\min} = s_{\max} = s_X(a)$, i.e., a and c are strongly connected in G. But this implies that a, c and v are strongly connected in G, which contradicts the fact that the SCC of v in G does not contain vertices of $\partial\mathcal{R}$. ◀

The running time analyses of both the update and query procedures are identical to the analyses of Section 3. Hence, we have proved the following theorem.

▶ **Theorem 2.** *Let G be a planar digraph. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions and deletions with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute the identifier of the strongly connected component of any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

References

1. 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*, pages 477–486, 2016. doi:10.1109/FOCS.2016.58.
2. 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*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
3. 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 *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 740–753, 2016. doi:10.1137/1.9781611974331.ch53.
4. Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1199–1218, 2012. doi:10.1145/2213977.2214084.
5. Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.

6 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 12–21, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320178.

7 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007. doi:10.1016/j.jalgor.2004.08.004.

8 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22, 2016. doi:10.1145/2756553.

9 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.

10 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.

11 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the o(mn) bound. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 389–397, 2016. doi:10.1145/2897518.2897521.

12 Aaron Bernstein and Shiri Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 453–469, 2017. doi:10.1137/1.9781611974782.29.

13 Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. *CoRR*, abs/2004.04496, 2020. arXiv:2004.04496.

14 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.

15 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *CoRR*, abs/2004.08432, 2020. arXiv:2004.08432.

16 Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.

17 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

18 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015. doi:10.1145/2684068.

19 Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012. doi:10.1007/s00453-010-9459-0.

20 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.

21 Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013. doi:10.1137/120864271.

22 Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. *Algorithmica*, 81(8):3075–3098, 2019. doi:10.1007/s00453-019-00570-z.

23 Panagiotis Charalampopoulos, Pawel 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, STOC 2019*, pages 138–151, 2019. doi:10.1145/3313276.3316316.

24 Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. Exact distance oracles for planar graphs with failing vertices. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2110–2123, 2019. doi:10.1137/1.9781611975482.127.

25 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 1–10, 2015. doi:10.1145/2746539.2746562.

26 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 170–181. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00025.

27 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC 2000*, pages 469–478, 2000. doi:10.1145/335305.335359.

28 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019. arXiv:1910.08025.

29 Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 389–400. ACM, 2019. doi:10.1145/3313276.3316320.

30 Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 962–973, 2017. doi:10.1109/FOCS.2017.93.

31 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

32 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Algorithms - ESA 2007, 15th Annual European Symposium, Proceedings*, pages 594–604, 2007. doi:10.1007/978-3-540-75520-3_53.

33 Hristo Djidjev. On-line algorithms for shortest path problems on planar digraphs. In *Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96*, pages 151–165, 1996. doi:10.1007/3-540-62559-3_14.

34 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

35 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. doi:10.1016/0196-6774(92)90004-V.

36 Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1319–1332, 2018. doi:10.1145/3188745.3188904.

37 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

38 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. doi:10.1016/j.jcss.2005.05.007.

39 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

40 Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 61:1–61:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.61.

41 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 515–529, 2018. doi:10.1137/1.9781611975031.34.

42 Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\epsilon)$ -approximate distance oracle for planar graphs. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2015. doi:10.1007/978-3-662-48971-0_53.

43 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 153–166. ACM, 2020. doi:10.1145/3357713.3384236.

44 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2542–2561. SIAM, 2020. doi:10.1137/1.9781611975994.155.

45 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2522–2541. SIAM, 2020. doi:10.1137/1.9781611975994.154.

46 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2562–2574. SIAM, 2020. doi:10.1137/1.9781611975994.156.

47 Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert Endre Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms*, 8(1):3:1–3:33, 2012. doi:10.1145/2071379.2071382.

48 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 146–155, 2014. doi:10.1109/FOCS.2014.24.

49 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Symposium on Theory of Computing, STOC 2014*, pages 674–683, 2014. doi:10.1145/2591796.2591869.

50 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. doi:10.1137/140957299.

51 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

52 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log\, n(\log\, \log\, n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 510–520, 2017. doi:10.1137/1.9781611974782.32.

53 Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.

54 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

55 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

56 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. doi:10.1145/3039873.

57 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

58 Adam Karczmarz. Decremental transitive closure and shortest paths for planar digraphs and beyond. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 73–92. SIAM, 2018. doi:10.1137/1.9781611975031.5.

59 Adam Karczmarz and Jakub Lacki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020*, pages 106–120. SIAM, 2020. doi:10.1137/1.9781611976014.15.

60 Adam Karczmarz and Jakub Łącki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *27th Annual European Symposium on Algorithms, ESA 2019*, pages 65:1–65:15, 2019. doi:10.4230/LIPIcs.ESA.2019.65.

61 Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 550–563, 2013. doi:10.1137/1.9781611973105.40.

62 Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms,SODA 2002*, pages 820–827, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545488.

63 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

64 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

65 Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n\log^2 n)$-time algorithm. *ACM Trans. Algorithms*, 6(2):30:1–30:18, 2010. doi:10.1145/1721837.1721846.

66 Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998. doi:10.1007/PL00009223.

67 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 121–130. ACM, 2010. doi:10.1145/1806689.1806708.

68 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, STOC 1984*, pages 376–382, 1984. doi:10.1145/800057.808703.

69 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 209–222, 2012. doi:10.1137/1.9781611973099.19.

70 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n\log^2 n/\log\log n)$ time. In *Algorithms - ESA 2010, 18th Annual European Symposium. Proceedings, Part II*, pages 206–217, 2010. doi:10.1007/978-3-642-15781-3_18.

71 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 950–961. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.92.

72 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.

73 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

74 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Proceedings*, pages 461–470, 2005. doi:10.1007/11533719_47.

75 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 372–383, 1993. doi:10.1007/3-540-57273-2_72.

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

77 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Proceedings*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8_33.

78 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC 2005*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

79 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

80 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 436–455. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00035.

81 Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, abs/1510.04590, 2015. arXiv:1510.04590.

82 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 539–549. SIAM, 2013. doi:10.1137/1.9781611973105.39.

83 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1757–1769. SIAM, 2013. doi:10.1137/1.9781611973105.126.

84 Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 351–362. SIAM, 2016. doi:10.1137/1.9781611974331.ch26.

A Omitted Proofs

▶ **Lemma 16.** *Let $\mathcal{Q}$ be some collection of pieces from $\mathcal{T}(G)$ such that:*
1. *for each leaf piece $L \in \mathcal{T}(G)$, either L or some ancestor of L is in $\mathcal{Q}$,*
2. *for each $H \in \mathcal{Q}$, if some ancestor of H is in $\mathcal{Q}$, then the parent of H is also in $\mathcal{Q}$.*

Then for any $u, v \in V(\mathrm{DDG}(\mathcal{Q}))$, $dist_{\mathrm{DDG}(\mathcal{Q})}(u, v) = dist_G(u, v)$.

Proof. Let us note that if for some $H \in \mathcal{T}(G)$ no ancestor of H belongs to $\mathcal{Q}$, then $\partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. We prove this claim by induction on the level ℓ of piece H. For $\ell = 0$, we get $H \in \mathcal{Q}$, so clearly $\partial H \subseteq V(H) \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. Suppose $\ell \geq 1$. The statement is trivial if $H \in \mathcal{Q}$. Otherwise, consider the children $H_1, \ldots, H_k$ of H. By induction we get $\partial H_i \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. So in fact we have $\partial H \subseteq \bigcup_{i=1}^{k} \partial H_i \subseteq V(\mathrm{DDG}(\mathcal{Q}))$.

Since the edges of each DDG_H encode lengths of some paths in G, we have that $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u, v) \geq \mathrm{dist}_G(u, v)$. We now prove that there is a path of length at most $\mathrm{dist}_G(u, v)$ in $\mathrm{DDG}(\mathcal{Q})$. Let P be some shortest $u \to v$ path in G. Let H be a piece of $\mathcal{T}(G)$ of minimum level that contains P. We prove our claim by induction on the level ℓ of H.

First note that by property 1 of $\mathcal{Q}$, if $H \notin \mathcal{Q}$ and no descendant of H belongs to $\mathcal{Q}$, then H has a nearest ancestor H^* such that $H^* \in \mathcal{Q}$. Observe that $V(\mathrm{DDG}(\mathcal{Q})) \cap V(H) \subseteq \partial H^*$. Hence, $u, v \in \partial H^*$ and thus $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_{\mathrm{DDG}_{H^*}}(u,v) = \mathrm{dist}_{H^*}(u,v) \leq \mathrm{dist}_H(u,v) = \mathrm{dist}_G(u,v)$.

So we can assume that either $H \in \mathcal{Q}$ or some descendant of H belongs to $\mathcal{Q}$. Then by property 2 of $\mathcal{Q}$, we have that either $H \in \mathcal{Q}$ or no ancestor of H belongs to $\mathcal{Q}$. In either of these cases we have $\partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. Suppose first $\ell = 0$ – then H is a leaf piece and thus $H \in \mathcal{Q}$. So clearly $P \subseteq H \subseteq \mathrm{DDG}(\mathcal{Q})$, i.e., $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_G(u,v)$. On the other hand, if we assume $\ell \geq 1$ then we can split P into maximal subpaths $P_1, \ldots, P_k$ such that each $P_i = u_i \to v_i$ is entirely contained in a single child of H_P. Then for each i we have $\{u_i, v_i\} \subseteq \{u, v\} \cup \partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$ so by induction we get that $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u_i, v_i) \leq \mathrm{dist}_G(u_i, v_i)$ which implies $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_G(u,v)$. ◀

▶ **Lemma 4** ([24, 38]). *Let $\mathcal{L}$ be some collection of leaf pieces of $\mathcal{T}(G)$. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L})))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L}))}(u,v)$.*
2. $\sum_{H \in \mathrm{cone}_G(\mathcal{L})} |\partial H| = O\left(\sqrt{n|\mathcal{L}|}\right)$.

Proof. To obtain item 1 it is enough to note that $\mathrm{cone}_G(\mathcal{L})$ satisfies the requirements posed on the collection $\mathcal{Q}$ in Lemma 16.

Let $\mathcal{A}$ be the set containing all ancestors of all the leaf pieces $L \in \mathcal{L}$. We show item 2 by bounding the sum $X = \sum_{H \in \mathcal{A}} |\partial H|$. Since the number of boundary vertices of a piece is bounded by the sum of numbers of boundary vertices of its parent and its sibling, the sum $\sum_{H \in \mathrm{cone}_G(\mathcal{L})} |\partial H|$ of our interest can be larger from X only by a constant factor.

Recall that $\mathcal{T}(G)$ admits an r-division for any $r \in [1, n]$, i.e., there exists such r-division $\mathcal{R}$ that $\mathcal{R} \subseteq \mathcal{T}(G)$ and the boundary of each piece of $\mathcal{R}$ equals the boundary of that piece in $\mathcal{T}(G)$. Let us split $\mathcal{A}$ into two parts: let $\mathcal{A}_1$ contain those $H \in \mathcal{A}$ that are descendants of some piece $P \in \mathcal{R}$, and let $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$. Let $X_i = \sum_{H \in \mathcal{A}_i} |\partial H|$.

Since each $L \in \mathcal{L}$ is a descendant of a unique piece $P \in \mathcal{R}$, we now bound the sum of $|\partial H|$ over all ancestors of L that are descendants of P. Recall (Section 2) that if H is a piece in a recursive decomposition of an n-vertex graph, then $|\partial H| = O(\sqrt{n}/c^d)$ where d is the depth of H in that decomposition. Consider the subtree of $\mathcal{T}(G)$ rooted at P – it forms a recursive decomposition $\mathcal{T}(P)$ of P with some initial boundary vertex set $|\partial P|$ of size $O(\sqrt{r})$. So the sum of $|\partial H|$ over all ancestors of L that are descendants of P is actually equal to the sum of $|\partial H|$ over all ancestors H of L in $\mathcal{T}(P)$. Since each ancestor has distinct integral depth, this sum is $O(\sum_{i=0}^{\infty} \sqrt{|P|}/c^i) = O(\sqrt{|P|}) = O(\sqrt{r})$. Hence $X_1 = O(|\mathcal{L}| \cdot \sqrt{r})$. On the other hand, $X_2 \leq \sum_{P \in \mathcal{R}} |\partial P| = O(n/\sqrt{r})$. So we obtain $X = X_1 + X_2 = O(n/\sqrt{r} + |\mathcal{L}| \cdot \sqrt{r})$. By choosing $r = n/|\mathcal{L}|$, we obtain $X = O(\sqrt{n \cdot |\mathcal{L}|})$ as desired. ◀

▶ **Lemma 5** ([24, 38]). *Let $X \subseteq V(G)$ be non-empty. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X)))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X))}(u,v)$.*
2. $\sum_{H \in \mathrm{cone}_{\mathcal{R}}(X)} |\partial H| = O\left(n/\sqrt{r} + \min\left(\sqrt{n \cdot |X|}, |X| \cdot \sqrt{r}\right)\right)$.

Proof sketch. The proof is completely analogous to that of items 2 and 3 of Lemma 4. It is enough to glue the individual decompositions $\mathcal{T}(P)$ into a single decomposition $\mathcal{T}'(G)$ such that the root has $O(n/r)$ children instead of just 2: the individual pieces of $\mathcal{R}$. Then item 1 follows by Lemma 16. Let $Y = \sum_{H \in \mathrm{cone}_{\mathcal{R}}(X)} |\partial H|$. Note that $Y = O(n/\sqrt{r} + |X| \cdot \sqrt{r})$. If we have $r = n/|X| \leq r$, then we can obtain the bound $Y = O(\sqrt{n \cdot |X|})$ in the proof of Lemma 4. Otherwise, $|X| \leq n/r$, so $Y = O(n/\sqrt{r} + |X| \cdot \sqrt{r}) = O(n/\sqrt{r})$. So $Y = O(n/\sqrt{r} + \sqrt{n \cdot |X|})$ in all cases as well. ◀

▶ **Theorem 6.** *Let $G = (V, E)$ be a weighted planar graph. Suppose that adding infinite-weight edges to G does not have effect on any properties of G that we care about. Let $r \in [1, n]$.*

There is a data structure maintaining an r-division with few holes $\mathcal{R}$ of some G^+ such that:

1. *G^+ is obtained from G by adding infinite-weight edges.*
2. *Each P has all its faces except its holes triangular and is accompanied with some auxiliary data structures that can be constructed in $T(r)$ time given P and use $S(r)$ space.*

The data structure uses $O\left(n + \frac{n}{r} \cdot S(r)\right)$ space and can be initialized in $O\left(n + \frac{n}{r} \cdot T(r)\right)$ time. After each edge deletion and edge insertion (preserving the planarity of G), it can be updated in $O(r + T(r))$ worst-case *time.*

Proof. On initialization, we first connect and triangulate G using infinite-weight edges, thus obtaining G^+. Then, we compute an r-division $\mathcal{R}$ with few holes of G^+ in linear time [64] and subsequently initialize the auxiliary data structures. Let $h \geq 2$ and $c \geq 8$ be constants such that a single piece of the computed r-division has at most $c\sqrt{r}$ boundary vertices distributed over h holes.

We will guarantee at all times that for any single piece P, $|\partial P| \leq 3c\sqrt{r}$, and there exist at most $3h$ faces of P such that any $v \in \partial P$ lies on one of these faces, called holes of P. Moreover, each edge of G^+ is contained in at most two pieces of $\mathcal{R}$: this is satisfied initially since the r-division of [64] forms a partition of faces of G^+.

Suppose that the removal of an edge $e = uv$ is issued to G. We then remove e from each of the at most two pieces P containing it. If P is disconnected afterwards, we replace it with two connected pieces P_1, P_2. Otherwise, since removing e merges two faces of P, the total number of holes of P does not increase. If, on the other hand, a new edge $e = uv$ is inserted, we add a new piece P_e consisting of a single edge e to $\mathcal{R}$. Adding P_e may cause an endpoint of e, say u, to become a boundary vertex of $\mathcal{R}$. If u was not a boundary vertex before the insertion, it had to be a vertex of a single piece P_u. At this point, since a new boundary vertex emerges in P_u, we might have $\partial P_u > 3c\sqrt{r}$ or ∂P might no longer lie on at most $3h$ faces of P_u. However, there surely exist some $3h + 1$ faces whose vertices include the whole set ∂P_u, and $|\partial P_u| \leq 3c\sqrt{r} + 1$. To fix our invariants, we first compute a cycle separator C of P_u wrt. the boundary vertices of P_u in $O(r)$ time, and replace P_u with two pieces $P_{u,1}, P_{u,2}$ – the two subgraphs of P_u induced by vertices weakly on one side of C. Clearly, the vertices of C become new boundary vertices afterwards. Subsequently, we similarly break each of $P_{u,1}, P_{u,2}$ further into two parts using a cycle separator wrt. the holes of this piece (see [64] for details). Each of the at most four resulting pieces has at most $\frac{2}{3}(3c\sqrt{r} + 1) + 2\sqrt{2}\sqrt{r} \leq (2c + 2\sqrt{2} + 2)\sqrt{r} \leq 3c\sqrt{r}$ boundary vertices, and at most $\frac{2}{3} \cdot (3h + 1) + 1 \leq 2h + 2 \leq 3h$ holes.

Observe that a single edge update can introduce $O(1)$ new pieces of size $O(r)$ in the maintained r-division, and the sizes of the existing pieces do not increase. For each of the affected pieces we recompute the auxiliary data structures in $O(T(r))$ time. As a result, after $O(n/r)$ updates, there are still $O(n/r)$ pieces, each of size $O(r)$ and with $O(\sqrt{r})$ boundary vertices distributed over $O(1)$ holes of that piece. Consequently, after every $\Omega(n/r)$ updates, we reinitialize $\mathcal{R}$ for the current graph G in $O\left(n + \frac{n}{r} \cdot T(r)\right)$ time. Hence, the amortized time to update $\mathcal{R}$ is $O(r + T(r))$.

Finally, observe that our data structure can be modified in a standard way (see e.g., [5]) to have $O(r + T(r))$ *worst-case* update time bound instead of just an amortized one. This is possible since our update procedure actually takes $O(r + T(r))$ worst-case time apart from once every $k = \Omega(n/t)$ updates when the whole data structure is rebuilt in $O\left(n + \frac{n}{r} \cdot T(r)\right)$

worst-case time. To this end we apply the time-slicing technique. We use two copies of our data structure switching their roles every $k/2$ updates. One copy is for handling at most $k/2$ updates and answering queries, and another is being gradually reinitialized in chunks of $\Omega(r + T(r))$ time (of either initialization or updates replayed) in the background. ◀

B Empty Voronoi Cells

Here, we briefly explain why the algorithm of [23] that underlies Theorem 7 works irrespective of whether there are empty Voronoi cells.

We need a few more definitions. Let f^* be the vertex of the dual graph of G corresponding to the face f, where the sites lie. For an additively weighted Voronoi diagram $\mathsf{VD}(S, \omega)$ over a triangulated graph (possibly apart from face f), we call a face whose incident vertices belong to three different Voronoi cells *trichromatic*. Let $\mathsf{VD}_0^*(S, \omega)$ be the forest obtained by considering the dual edges of G whose endpoints belong to different Voronoi cells. Then, we obtain $\mathsf{VD}_1^*(S, \omega)$ by repeatedly contracting edges of the forest that have an endpoint of degree 2. We obtain $\mathsf{VD}_2^*(S, \omega)$ by creating one copy of f^* for each of its incident edges – inheriting only that edge. The vertices of $\mathsf{VD}_2^*(S, \omega)$ that are not copies of f^* are in one-to-one correspondence with the trichromatic faces of G, other than f. If $\mathsf{VD}_2^*(S, \omega)$ is a tree then we are done. Otherwise, is some cell of $\mathsf{VD}(S, \omega)$ is empty, $\mathsf{VD}_2^*(S, \omega)$ might be a forest. However, as shown in Section 6 of [41], $\mathsf{VD}_2^*(S, \omega)$ can be turned into the sought tree $\mathsf{VD}^*(S, \omega)$ in $O(|S|)$ time.

The algorithm of [23] for computing $\mathsf{VD}^*(S, \omega)$ implicitly assumes that all Voronoi cells are non-empty, and hence that $\mathsf{VD}_2^*(S, \omega)$ is a tree. This algorithm performs FR-Dijkstra computations on G in order to compute the trichromatic faces of G, and a representation of shortest paths from the sites to the vertices incident to these trichromatic faces. Then, $\mathsf{VD}_2^*(S, \omega)$ can be straightforwardly retrieved from this representation. If we run the same algorithm without assuming that there are no non-empty Voronoi cells, the sites with empty Voronoi cells can be easily read from the shortest paths tree obtained from FR-Dijkstra (more specifically, these are the sites whose some ancestor in this tree is also in S). Since the proof of correctness of the algorithm of [23] for computing the trichromatic faces actually only requires the set S to lie on a single face of G (and not necessarily to be equal to $V(f)$), we can re-run it with S pruned from "empty" sites. From all the trichromatic faces and the corresponding shortest paths from sites to their incident vertices, we can retrieve the forest $\mathsf{VD}_2^*(S, \omega)$ in the same way as if it were a tree. Finally, we can then turn this forest into the sought tree $\mathsf{VD}^*(S, \omega)$ in $O(|S|)$ time as in [41].

C Negative Edges in the Fully Dynamic SSSP Algorithm

Recall that $p : V(H) \to \mathbb{R}$ is called a feasible price function of H if $\mathrm{dist}_H(u, v) + p(u) - p(v) \geq 0$ for all $u, v \in V(H)$. A feasible price function is guaranteed to exist if H contains no negative-cost cycle. It is well-known that, provided that a graph H is strongly connected, a vector of distances from any vertex of H constitutes a feasible price function of H.

As in the fully dynamic all-pairs algorithm of [56], we maintain functions $\phi : \partial\mathcal{R} \to \mathbb{R}$ and $\phi_P : \partial P \to \mathbb{R}$, where $P \in \mathcal{R}$, such that ϕ is some feasible price function of G restricted to $\partial\mathcal{R}$, and each ϕ_P is a feasible price function of P.

Since single-source shortest paths in planar graphs with negative weights can be computed in $O(n \log^2 n)$ time [65,70], each ϕ_P can be seen as an accompanying data structure of piece P computable in $O(r \log^2 r)$ time and maintained by the fully dynamic r-division algorithm.

The functions ϕ_P allow to treat individual graphs P and P_h as non-negatively weighted when computing all the needed DDGs and MSSP data structures, and also point location data structures $L(P)$.

It is known [40, 56] that the FR-Dijkstra algorithm (as in Lemma 3) can handle negative weights in $\mathrm{DDG}(\mathcal{H})$ with no asymptotic overhead if a feasible price function on $\mathrm{DDG}(\mathcal{H})$ is provided. Therefore, we would like to have a feasible price function on $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ to compute distances from s in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ needed by the update algorithm. We can extend ϕ from ∂R to all vertices in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ as follows. Note that all pieces in $\mathrm{cone}_{\mathcal{R}}(s)$ except of those in $\mathcal{R}$ have their parents also in $\mathrm{cone}_{\mathcal{R}}(s)$. We call those pieces $H \in \mathrm{cone}_{\mathcal{R}}(s)$ for which we know the value of ϕ on all of ∂H *processed*. Initially, only the pieces $P \in \mathcal{R}$ are processed by the definition of ϕ. While there are still unprocessed pieces, we take any unprocessed piece H whose parent $A \in \mathcal{T}(P)$ has already been processed. Let H' be the sibling of H in $\mathcal{T}(P)$. Observe that ϕ_P is a feasible price function of A as well. We extend ϕ to boundary vertices of H, H' by computing shortest paths on $\mathrm{DDG}(\{H, H'\})$ from vertices ∂A, with the initial distance to each $v \in \partial A$ set to $\phi(v)$, and using FR-Dijkstra (Lemma 3) with price function ϕ_P. This way, only the initial distances of ∂A are possibly negative from the point of view of FR-Dijkstra. This does not constitute a problem for either Dijkstra's algorithm or FR-Dijkstra though (see [56]; one can treat the initial distances as weights of edges going out of a super-source; these weights can be all increased by the same large value to be made positive). One can show from the definition of ϕ that the values of ϕ on ∂A will not be altered and the computed distances form a feasible price function on $\partial H \cup \partial H'$ in G. Hence, given that $\partial A \subseteq \partial H \cup \partial H'$, we can process the children H, H' of a processed piece A in $O((|\partial H| + |\partial H'|) \log^2 n)$ time. Summing over all pieces, we obtain by Lemma 5 that extending ϕ to all $V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s)))$ takes $O(n/\sqrt{r} \log^2 n)$ time. This cost is therefore negligible.

Note that an edge deletion or weight increase cannot break the feasibility of ϕ. We might need to recompute ϕ only upon insertion or weight decrease of some edge uv. As shown by Kaplan et al. [56], the new "global" price function ϕ' (or a negative cycle) can be found by computing distances from v to $\partial\mathcal{R} \cup \{u\}$ in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(u, v))$ (before applying the insertion) using FR-Dijsktra and the old price function ϕ. This can be done in $O(n/\sqrt{r} \log^2 n)$ time by first extending the old ϕ to $V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(u, v)))$ as described above and then running the single-source shortest paths algorithm of Lemma 3.