Fully Dynamic Algorithms for Minimum Weight Cycle and Related Problems

Adam Karczmarz
Institute of Informatics, University of Warsaw, Poland

Abstract

We consider the directed minimum weight cycle problem in the fully dynamic setting. To the best of our knowledge, so far no fully dynamic algorithms have been designed specifically for the minimum weight cycle problem in general digraphs. One can achieve $\widetilde{O}(n^2)$ amortized update time by simply invoking the fully dynamic APSP algorithm of Demetrescu and Italiano [J. ACM '04]. This bound, however, yields no improvement over the trivial recompute-from-scratch algorithm for sparse graphs.

Our first contribution is a very simple deterministic $(1+\epsilon)$-approximate algorithm supporting vertex updates (i.e., changing all edges incident to a specified vertex) in conditionally near-optimal $\widetilde{O}(m\log(W)/\epsilon)$ amortized time for digraphs with real edge weights in $[1, W]$. Using known techniques, the algorithm can be implemented on planar graphs and also gives some new sublinear fully dynamic algorithms maintaining approximate cuts and flows in planar digraphs.

Additionally, we show a Monte Carlo randomized exact fully dynamic minimum weight cycle algorithm with $\widetilde{O}(mn^{2/3})$ worst-case update that works for real edge weights. To this end, we generalize the exact fully dynamic APSP data structure of Abraham et al. [SODA'17] to solve the *multiple-pairs shortest paths* problem, where one is interested in computing distances for some k (instead of all n^2) fixed source-target pairs after each update. We show that in such a scenario, $\widetilde{O}((m+k)n^{2/3})$ worst-case update time is possible.

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

Keywords and phrases Dynamic graph algorithms, minimum weight cycle, dynamic shortest paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.83

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2106.11744

Funding *Adam Karczmarz*: supported by ERC Consolidator Grant 772346 TUgbOAT and by the Foundation for Polish Science (FNP) via the START programme.

1 Introduction

The all-pairs shortest paths problem (APSP) is one of the most fundamental graph problems. Given a real-weighted *directed* graph G with n vertices, the goal is to compute the distance matrix between all pairs of vertices u, v in G. APSP can be computed in $\widetilde{O}(nm)$ time [29, 41], which is clearly near-optimal for sparse graphs (since the output consists of n^2 numbers), but is also conjectured to be optimal for the entire range of possible graph sparsities. Some of the other core directed graph problems such as computing the diameter, the radius, or the minimum weight cycle[1] can be trivially reduced to APSP in $O(n^2)$ time by simply inspecting the entries of the distance matrix. In fact, as shown by Vassilevska Williams and Williams [47], for dense graphs APSP is known to be subcubically equivalent to many problems which look easier at first sight, especially because their output is just a single

[1] Also called the *girth*, or the *weighted girth* of a digraph. For simplicity, in this paper we very often use *minimum weight cycle* to refer to the *length* of such a cycle rather than to the actual cycle. Moreover, throughout this paper, our focus is on computing/maintaining that length instead of the actual cycle. The obtained algorithms, however, can be easily extended to return a sought cycle with no additional asymptotic overhead.

number (as opposed to n^2 numbers in APSP). These include e.g., the radius, the minimum weight cycle, and the second shortest simple s, t path problems. For all these problems, just like for APSP, the best known algorithms run in $\widetilde{O}(nm)$ time. Lincoln et al. [36] gave some compelling reasons why improving upon this bound may also be impossible.

In this paper, our focus is on *fully dynamic* graph algorithms. Fully dynamic graph algorithms allow updating the graph under both edge insertions and deletions, as opposed to *partially dynamic* algorithms that allow either only insertions (*incremental* setting) or only deletions (*decremental* setting). Fully dynamic APSP has been widely studied in the past. Demetrescu and Italiano [18] showed that the distance matrix can be *explicitly* maintained in $\widetilde{O}(n^2)$ *amortized* time under *vertex updates* which are allowed to change all edges incident to a single vertex at once. Thorup [45] simplified and slightly improved their algorithm. Clearly, if the algorithm is required to maintain all distances explicitly, one cannot break through the $O(n^2)$ time barrier since even a *single* edge update may change *all* the n^2 pairwise distances. Much of the work in this topic [2, 24, 46] has been devoted to obtaining good *worst-case* bounds on the time needed to recompute the distance matrix and it is known that $\widetilde{O}(n^{2+2/3})$ worst-case update time is possible [2, 24]. Interestingly, none of the known fully dynamic algorithms for *real-weighted* dynamic APSP has $o(n^2)$ update time and a non-trivial query procedure running in $o(m)$ time. Such an algorithm, with $\widetilde{O}(m\sqrt{n})$ amortized update time and $\widetilde{O}(n^{3/4})$ query time, has so far been only described for sparse enough unweighted graphs by Roditty and Zwick [43].

The algorithm of Demetrescu and Italiano [18] immediately implies $\widetilde{O}(n^2)$ amortized update bound for fully dynamic variants of all the most fundamental problems "trivially reducible" to APSP – the aforementioned diameter, radius, or minimum weight cycle. Surprisingly, as shown in [3], such an update bound is likely to be the best possible for maintaining both the diameter and the radius (conditionally on so-called Strong Exponential Time- and Hitting Set hypotheses [1, 26]), even if the graph remains sparse at all times and $(3/2 - \epsilon)$-approximation is allowed.

It is thus natural to ask whether there exist fully dynamic algorithms for the *minimum weight cycle problem* that improve upon the reduction to fully dynamic APSP for sparse graphs, possibly allowing some small multiplicative approximation. The fundamental difference between the minimum weight cycle and diameter/radius problems is that the trivial reduction of minimum weight cycle requires reading only m entries of the distance matrix, as opposed to all n^2 in the case of radius and diameter. As a result, by using the aforementioned fully dynamic algorithm of Roditty and Zwick [43], one immediately gets $\widetilde{O}(mn^{3/4})$ amortized update bound but merely for *unweighted* graphs. Note that this bound is always better than recompute-from-scratch, and is truly subquadratic for sparse graphs. It is however not clear whether such a bound can be obtained for real-weighted graphs, nor whether a much better bound is attainable if we allow approximation.

Motivated by the above, in this paper we initiate the study of the directed minimum weight cycle problem in the fully dynamic setting. To the best of our knowledge, this problem has not been explicitly studied in the literature before. It is worth noting, however, that a non-trivial fully dynamic algorithm has been shown for *undirected planar* graphs [37].

1.1 Our results

A fully dynamic approximate minimum weight cycle algorithm. Our first contribution is a simple deterministic fully dynamic algorithm maintaining a $(1+\epsilon)$-approximation of the minimum weight $\phi(G)$ of a cycle in a real-weighted directed graph G. If G has a negative cycle, then we define $\phi(G)=-\infty$, thus allowing the sought cycle to be non-simple. Note that if we wanted the minimum weight cycle to be simple and simultaneously allowed negative edge weights, the problem would become NP-hard via a reduction from Hamiltonian cycle.

▶ **Theorem 1.** *Let G be an initially empty fully dynamic real-weighted digraph such that the weight of each positive weight cycle in G always belongs to the interval $[c,C]$, $c,C\in\mathbb{R}$.*

There exists an algorithm maintaining an estimate ϕ' satisfying $\phi(G)\le\phi'\le(1+\epsilon)\phi(G)$ under vertex updates to G with amortized update time $O((m+n\log n)\cdot\log(C/c)/\epsilon)$.

By Theorem 1, a simpler amortized update time bound of $O((m+n\log n)\cdot\log(nW)/\epsilon)$ for the fully dynamic $(1+\epsilon)$-approximate minimum weight cycle problem can be obtained in two special cases:

- if G has real-weights in $\{0\}\cup[1,W]$,
- if G has integer weights in $(-\infty,W]$.

Via known conditional lower bounds on the static approximate minimum weight cycle problem, the update time bound in Theorem 1 – as a function of m *alone* – is near-optimal for both vertex and edge updates if we allow approximation factor less than 2 and $O(m^{2-\delta})$ preprocessing time (for some $\delta>0$). Indeed, Dalirrooyfard and Vassilevska Williams [17] proved that under so-called k-Cycle hypothesis [3], one cannot approximate the minimum weight cycle within factor less than 2 in $O(m^{2-\delta})$ time, for any $\delta>0$. Clearly, if there was, say, a dynamic 3/2-approximate minimum weight cycle algorithm with $O(m^{2-\delta})$ preprocessing time, $O(m^{1-\delta})$ update time, and the same interface as our algorithm, m edge/vertex updates would be sufficient to obtain a static 3/2-approximate minimum weight cycle algorithm running in $O(m^{2-\delta})$ time. This would refute the k-Cycle hypothesis.

Observe that the $\Omega(m^{2-o(1)})$ conditional lower bound [17] (which implies that the $\Omega(mn^{1-o(1)})$ bound holds for *some* sparsity m) on the complexity of static approximate minimum weight cycle problem does not rule out dynamic vertex update bounds of the form $\widetilde{O}(n^{\alpha}\cdot m^{1-\alpha})$ for some $\alpha\in(0,1]$ or $\widetilde{O}(m^{1+\beta}/n^{2\beta})$ for some $\beta\in(0,1/2]$, e.g., $\widetilde{O}(n)$, $\widetilde{O}(\sqrt{nm})$, or $\widetilde{O}(m^{3/2}/n)$. However, if we limit ourselves to *"combinatorial"* algorithms that do not rely on fast matrix multiplication, such $O(m^{1-\epsilon})$ bounds are ruled out for infinitely many sparsities of the form $m=\Theta(n^{1+2/(k-1)})$, where $k\ge 3$ is an odd integer [17, 36].

We stress that the aforementioned static conditional lower bounds do not rule out $\widetilde{O}(n)$ or even $\widetilde{O}(\sqrt{nm})$ amortized update time in the *edge update* model. In this case, for similar reasons, only combinatorial approximate algorithms with amortized update time that is sublinear in n for many sparsities, e.g., $\widetilde{O}(m^{\beta}\cdot n^{1-2\beta})$ for $\beta\in(0,1/2]$, are unlikely to exist.

Fully dynamic cycles, flows, and cuts in planar graph. Interestingly, if we limit our attention to the case of single edge updates (as opposed to vertex updates) and real weights in $\{0\}\cup[1,W]$, the amortized update cost of the data structure of Theorem 1 can always be charged to the cost of performing a single edge update plus a single distance query on $O(\log(nW)/\epsilon)$ fully dynamic *exact* distance oracles, each maintaining some subgraph of G. For general digraphs, this amounts to running Dijkstra's algorithm in each of these subgraphs since no non-trivial fully dynamic distance oracles with both update and query time $o(m)$ are known. However, such dynamic distance oracles are well-known to exist for planar digraphs [21, 31, 34] which immediately leads to the following result.

▶ **Theorem 2.** *Let G be a planar digraph G with real weights in $\{0\}\cup[1,W]$. There exists an algorithm maintaining an $(1+\epsilon)$-approximate estimate of $\phi(G)$ under planarity preserving edge insertions and deletions with amortized update time $\widetilde{O}(n^{2/3}\log(W)/\epsilon)$.*

Previously, no sublinear fully dynamic algorithm for minimum weight cycle in planar directed graphs has been described. An *exact* algorithm for planar *undirected* graphs with $\widetilde{O}(n^{5/6})$ update time was given by Łącki and Sankowski [37].

There is a well-known correspondence between simple cuts in an undirected plane graph G, and simple cycles in its dual G^*. The correspondence, in a way, extends to *directed* planar graphs (see e.g. [35, 38]). Nevertheless, currently the best known min s,t-cut algorithms in planar digraphs [9, 19] are less efficient and use entirely different techniques than their counterparts for planar undirected graphs [28]. Generally speaking, for cut/flow applications, undirected planar graphs proved much more friendly to work with (see e.g., the discussion in [19] or [38]). As an example of this phenomenon, an *exact* fully dynamic max s,t-flow *oracle* (accepting s,t as query parameters) with $\widetilde{O}(n^{2/3})$ update and query time exists for undirected plane graphs [28], whereas no such dynamic algorithm has been described for *directed* plane graphs, even allowing approximation and just a single fixed source-sink pair.

It is known that in a plane digraph G, an s,t-flow of value f can be routed iff the dual $G^*_{s,t,f}$ of a certain augmentation of $G_{s,t,f}$ depending on s,t and f contains no negative cycles [19, 30, 39]. Roughly speaking, since the algorithm of Theorem 2 supports negative weights, by running it on $G^*_{s,t,f}$ for $O(\log(nW)/\epsilon)$ distinct values of f, we obtain the following.

▶ **Theorem 3.** *Let G be a plane embedded digraph with real edge capacities in $\{0\} \cup [1, W]$ and a* fixed *source/sink pair s,t. There exists an algorithm maintaining a $(1-\epsilon)$-approximate estimate of the value of maximum s,t-flow in G under* embedding preserving *edge insertions and deletions with $\widetilde{O}(n^{2/3}\log(W)/\epsilon)$ amortized update time.*

To the best of our knowledge, the above constitutes the first known fully dynamic maximum s,t-flow algorithm for plane directed graphs with a sublinear update time bound.

Exact fully dynamic minimum weight cycle and MPSP. Finally, we consider maintaining the minimum weight cycle *exactly* in a fully dynamic *real-weighted* digraph. We show:

▶ **Theorem 4.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic algorithm maintaining $\phi(G)$ under vertex updates with $O((m + n\log n)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*[2]

Note that for sparse graphs, Theorem 4 allows recomputing the minimum weight cycle in $\widetilde{O}(n^{5/3})$ time, i.e., polynomially faster than recompute-from-scratch and the dynamic algorithm of Demetrescu and Italiano [18]. However, observe that [18] yields a better amortized update bound for $m = \omega(n^{4/3})$.

In order to obtain Theorem 4, we generalize the fully dynamic APSP algorithm of Abraham et al. [2] in a non-trivial way to solve what we call the *multiple pairs shortest paths* problem (*MPSP*). In the MPSP problem, which may be of independent interest, one requires to maintain only k fixed entries of the distance matrix, i.e., after each update we are interested in distances between some source-target pairs (s_i, t_i) for $i = 1, \ldots, k$. Recall that the minimum weight cycle of a directed graph can be computed by inspecting distances for m source-target pairs. We obtain the following bound for the fully dynamic MPSP problem.

▶ **Theorem 5.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic MPSP data structure supporting vertex updates with $O((m+n\log n+k)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

[2] That is, with probability at least $1 - 1/n^c$ for any chosen constant $c \geq 1$.

Note that the aforementioned data structure of Roditty and Zwick [43] trivially implies an MPSP data structure for *unweighted* digraphs with $\widetilde{O}(m\sqrt{n}+kn^{3/4})$ amortized update bound. Our result shows that a better (even worst-case) bound for (even real-weighted) sparse graphs can be achieved if the set of source-target pairs is fixed throughout.

Actually, just as the worst-case update time of the data structure of Abraham et al. [2] can be very easily improved to $\widetilde{O}(n^{2.5})$ for *unweighted* graphs [2, Section 4.2], an unweighted variant of our MPSP data structure has $\widetilde{O}((m+k)\sqrt{n})$ worst-case update time.

Interestingly, it seems that the other known approaches to fully dynamic APSP in real-weighted graphs [18, 23, 45], if adjusted, cannot easily yield subquadratic (in n) update times for "sparse" instances of MPSP where $m, k = O(n)$. This is because they all reconstruct shortest paths in a hierarchical manner, by inductively stitching [23] or extending [18, 45] paths recomputed earlier in the process. Even though the number of input source-target pairs of interest may be small, these may require answers for $\Theta(n^2)$ distinct source-target pairs at lower levels of the hierarchy. The data structure of Abraham et al. [2], on the contrary, does not use a hierarchical approach and can be thought as using a single "stitching layer".

Since the algorithm behind Theorem 4 (Theorem 5) is exact, the maintained information, i.e., the minimum weight of a cycle (the entries of the distance matrix of interest, resp.) is unique. Therefore, if we are interested in maintaining the corresponding weight (distances, resp.) only, the bounds in Theorems 4 and 5 hold against an adaptive adversary. However, if we are required to output some actual minimum weight cycle (edges on some of the desired shortest paths, resp.) we have to assume an oblivious adversary.[3]

1.2 Related work

Computing minimum weight cycles statically. The best known algorithm for computing the minimum weight cycle in sparse graphs exactly runs in $O(nm)$ time [40]. One can improve upon this for graphs with small integer weights using matrix multiplication [16, 27, 42]. A subcubic-time $(1+\epsilon)$-approximation can also be achieved this way [10, 48]. Much of the recent work regarded approximating the minimum weight cycle within factor at least 2 [13, 14, 17].

Dynamic APSP. Apart from the fully dynamic setting, APSP has also been widely studied in partially dynamic settings. There exist efficient exact algorithms for *unweighted* digraphs with $\widetilde{O}(n^3)$ total update time in both incremental [4] and decremental [5, 20] settings. The fully dynamic APSP algorithm [18, 45] is known to have total update time $\widetilde{O}(n^3)$ in the decremental setting for real-weighted digraphs, but only when each update removes *all* edges incident to a vertex (and thus there are at most $\leq n$ updates). For weighted digraphs, a nearly optimal $\widetilde{O}(nm\log(W)/\epsilon)$ total update time partially dynamic algorithm is known in the $(1+\epsilon)$-approximate setting [7]. This algorithm assumes an oblivious adversary though. Less efficient algorithms that are either deterministic or assume an adaptive adversary are known [20, 33, 32]. Note that many of the above algorithms maintain the distance matrix explicitly so they can be obviously used to maintain the minimum weight cycle (possibly approximately) in the respective partially dynamic scenarios.

Dynamic APSP has also been studied in undirected graphs [6, 8, 12, 15, 23, 25, 44].

[3] Abraham et al. [2] show how to extend their data structure so that it is capable of tracking lexicographically smallest shortest paths and thus works against an adaptive adversary, even when returning actual paths is required. Out of the box, this additional feature costs $\Omega(n^2)$ extra time per update, though. Adapting this idea to minimum weight cycle and MPSP is an interesting possible further step.

1.3 Organization of the paper

The rest of this paper is organized as follows. In Section 2 we fix the notation. In Section 3 we show a fully dynamic *threshold cycle detection* data structure that constitutes the heart of the fully dynamic $(1+\epsilon)$-approximate minimum weight cycle algorithm of Theorem 1 proved in Section 4. The applications of Theorem 1 to planar graph algorithms, in particular the proofs of Theorems 2 and 3, are covered in detail in Section 5. In Section 6 we describe the exact fully dynamic minimum weight cycle and fully dynamic MPSP algorithms. Due to space constraints, Section 6 contains merely an overview of the adjustments we make to the fully dynamic APSP algorithm of [2], and the details can be found in the full version.

2 Preliminaries

In this paper we deal with *real-weighted directed* graphs. We write $V(G)$ and $E(G)$ to denote the sets of vertices and edges of G, respectively. We denote by n and m numbers of vertices and edges (resp.) in the input graph. A graph H is a *subgraph* of G, which we denote by $H \subseteq G$, if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We write $uv \in E(G)$ when referring to edges of G and use $w_G(uv)$ to denote the weight of uv.

For an edge set F, we sometimes write $G+F$ to denote the graph $(V(G), E(G) \cup F)$. If F contains an edge uv of weight x and $uv \in E(G)$, then we assume that $w_{G+F}(uv) = \min(w_G(uv), x)$. For an edge e we sometimes use $G+e$ to denote $G+\{e\}$. For a subset $D \subseteq V$, we define $G \setminus D$ to be the graph G with all edges incident to vertices in D removed.

A sequence of edges $P = e_1 \dots e_k$, where $k \geq 1$ and $e_i = u_i v_i \in E(G)$, is called an $s \to t$ path in G if $s = u_1$, $v_k = t$ and $v_{i-1} = u_i$ for each $i = 2, \dots, k$. For brevity we sometimes also express P as a sequence of $k+1$ vertices $u_1 u_2 \dots u_k v_k$ or as a subgraph of G with vertices $\{u_1, \dots, u_k, v_k\}$ and edges $\{e_1, \dots, e_k\}$. A path P is *simple* if $u_i \neq u_j$ for $i \neq j$. A *cycle* is a path such that $u_1 = v_k$. A *simple cycle* is a cycle that is a simple path.

The *hop-length* of P is the number of edges in P. We also say that P is a *k-hop path*. The *length* of the path $\ell(P)$ is defined as $\ell(P) = \sum_{i=1}^{k} w_G(e_i)$. For convenience, we sometimes consider a single edge uv as a path of hop-length 1. If P_1 is a $u \to v$ path and P_2 is a $v \to w$ path, we denote by $P_1 \cdot P_2$ (or simply $P_1 P_2$) a path obtained by concatenating P_1 with P_2.

The *distance* $\delta_G(u,v)$ between the vertices $u, v \in V(G)$ is the length of the shortest $u \to v$ path in G, or ∞, if no $u \to v$ path exists in G.

Note that the distance is well-defined only if G contains no negative cycles. It is well known that G has no negative cycles if and only if there exists a *feasible price function* $p : V \to \mathbb{R}$ satisfying $w_G(e) + p(u) - p(v) \geq 0$ for all $uv = e \in E(G)$. It is well-known that, given a feasible price function of G, one can compute single-source shortest paths in G using Dijkstra's algorithm even if G has edges with negative weights.

Define $\phi(G)$ to be the infimum of $\ell(C)$ through all cycles $C \subseteq G$. Note that here C is not necessarily a simple cycle: in general finding minimum weight simple cycles with arbitrary negative weights is NP-hard. In particular, if G contains no cycles at all, then we define $\phi(G) := \infty$. If G contains a negative cycle, then $\phi(G) = -\infty$. On the other hand, if $\phi(G) \geq 0$, then G contains a simple cycle C' with $\ell(C') = \phi(G)$. We call any such cycle C' a *minimum weight cycle*. Observe that if $\phi(G) \geq 0$, then $\phi(G) = \min_{uv \in E(G)}\{\delta_G(v,u) + w_G(uv)\}$.

▶ **Observation 6.** *Let H be a non-negatively weighted digraph and let v be its vertex. The minimum weight of a cycle in H that goes through v can be computed in $O(m + n \log n)$ time.*

Proof. First compute single-source shortest paths from v using Dijkstra's algorithm. Note that the minimum weight cycle through v has length $\min_{uv \in E(H)}\{\delta_H(v,u) + w_H(uv)\}$. ◀

When characterizing dynamic graph algorithms, we use the term *edge update* to refer to a graph update that changes (i.e., inserts, removes, or alters the weight) a single edge of G. On the other hand, a *vertex update* can change all edges incident (incoming or outgoing) to a single chosen vertex $v \in V(G)$. In this case, we say that such a vertex update is *centered* at v.

3 Fully dynamic threshold cycle detection

Consider the following decision variant of the fully dynamic minimum weight cycle problem. Suppose we would like to maintain the information whether the minimum weight $\phi(G)$ of a cycle in a real-weighted digraph G is below some threshold $\mu \geq 0$. In this section we show:

▶ **Theorem 7.** *Let G be an initially empty real-weighted digraph and let $\mu \geq 0$. There exist a fully dynamic algorithm maintaining the information whether $\phi(G) < \mu$ and supporting vertex updates in $O(m + n \log n)$ amortized time.*

The idea is to keep the edge set E partitioned into two subsets E_0 and E_1 such that the following two invariants are satisfied:

(1) For $G_0 = (V, E_0)$ we have $\phi(G_0) \geq \mu$.
(2) If $E_1 \neq \emptyset$, then $\phi(G) < \mu$.

Observe that by the above invariants, $\phi(G) < \mu$ *if and only if* $E_1 \neq \emptyset$.

Let us first consider the case when G has non-negative edges only. Then we can assume $\mu > 0$ since the answer for $\mu = 0$ is trivially "no".

We store E_1 partitioned into subsets $E_1(v)$ for $v \in V$, so that each edge $uv \in E_1$ is stored in either $E_1(u)$ or $E_1(v)$ (this choice is arbitrary). Since the data structure is initialized with an empty graph, initially $E_0 = \emptyset$ and $E_1(v) = \emptyset$ for all $v \in V$.

We also store the vertices v with $E_1(v) \neq \emptyset$ of G in a list Q sorted by the time when the last insertion around v happened, i.e., at the end of Q we have a vertex that has been most recently subject to insertion of edges around v.

Let us now describe an auxiliary procedure `update`(v) that will be used to fix the invariants. `update`(v) does the following. We assume that $E_1(v) \neq \emptyset$. We compute the minimum weight x of a cycle going through v in $G_0 + E_1(v) = (V, E_0 \cup E_1(v))$ as described in Observation 6. If $x \geq \mu$, the edges $E_1(v)$ are moved to E_0, and the set $E_1(v)$ is emptied. This change is reflected in Q by removing v from Q.

To handle and insertion of a set F_v of edges centered at some vertex v, we simply add the edges F_v to $E_1(v)$, move v to the end of Q, and, if $Q = \{v\}$, run `update`(v).

To handle a deletion of an arbitrary set of edges $F \subseteq E$, we first remove each edge $f \in F$ from E_0 or some set $E_1(w)$, wherever f resides. If some $E_1(w)$ is emptied this way, w is removed from Q accordingly. Next, while $Q \neq \emptyset$, we repeatedly run `update`(v) for the first element $v \in Q$ and stop if `update`(v) fails to empty the respective set $E_1(v)$.

We now prove the correctness of the algorithm, whose pseudocode is given in Algorithm 1.

▶ **Observation 8.** *Suppose $\phi(G_0) \geq \mu$ and let $v \in V$. Then $\phi(G_0 + E_1(v)) < \mu$ if and only if the shortest cycle going through v in $G_0 + E_1(v)$ has weight less than μ.*

Proof. By $\phi(G_0) \geq \mu$, a cycle of weight less than μ in $G_0 + E_1(v)$ has to go through an edge of $E_1(v)$. All of these edges are incident to the vertex v. ◀

Clearly, E_0 and E_1 form a partition of E after each insertion or deletion: the procedure `update` only moves edges from E_1 to E_0.

Algorithm 1 Detecting a cycle of weight less than μ.

procedure `update`(v)
- $x :=$ the minimum weight of a cycle going through v in $G_0 + E_1(v)$
- **if** $x \geq \mu$ **then**
  - $E_0 := E_0 \cup E_1(v)$
  - $E_1(v) := \emptyset$
  - $Q := Q \setminus \{v\}$

procedure `insert`$(F_v \neq \emptyset)$
- $E_1(v) := E_1(v) \cup F_v$
- `move-to-back`(Q, v)
- **if** $Q = \{v\}$ **then**
  - `update`(v)

procedure `delete`$(F \subseteq E(G))$
- $E_0 := E_0 \setminus F$
- **for** $uv = e \in F$ **do**
  - **for** $w \in \{u, v\}$ **do**
    - $E_1(w) := E_1(w) \setminus \{e\}$
    - **if** $E_1(w) = \emptyset$ **then**
      - $Q := Q \setminus \{w\}$
- **while** $Q \neq \emptyset$ **do**
  - $v :=$ `front`(Q)
  - `update`(v)
  - **if** $E_1(v) \neq \emptyset$ **then**
    - **break**

function `cycle-below-threshold`$()$
- **return** $Q \neq \emptyset$

▶ **Lemma 9.** *Invariant (1) is maintained throughout the updates.*

Proof. Note that no edge is added to E_0 outside the `update` procedure. As a result, since invariant (1) cannot be broken by removing edges from G_0, to establish that invariant (1) is maintained, it is enough to see that `update` only adds edges to E_0 if $\phi(G_0) \geq \mu$ afterwards. ◀

▶ **Lemma 10.** *Invariant (2) is maintained throughout the updates.*

Proof. Let G', G'_0, E'_1 denote G, G_0, E_1 respectively *before* the graph update. Suppose that after processing the update, invariant (2) is broken. Equivalently, $E_1 \neq \emptyset$ and $\phi(G) \geq \mu$.

Suppose the update was insertion of edges F_v centered at v. Since adding edges can only decrease the minimum weight of a cycle, $\phi(G') \geq \mu$. As invariant (2) was satisfied before, $E'_1 = \emptyset$. So after F_v is moved to $E_1(v)$, we indeed have $Q = \{v\}$. Since $\phi(G_0 + E_1(v)) \geq \phi(G) \geq \mu$, $E_1(v)$ should have been moved to E_0 by `update`(v). But $E_1 = E_1 \setminus E'_1 \subseteq E_1(v) = \emptyset$, so $E_1 = \emptyset$, a contradiction.

Now assume that the update deleted an arbitrary subset of edges. If after some `update`(v) call we have $E_1(v) \neq \emptyset$, then $\phi(G_0 + E_1(v)) < \mu$, which implies $\phi(G) < \mu$, a contradiction. If no such v exists, then Q is emptied, i.e., $E_1(v) = \emptyset$ for all $v \in V$ after the deletion is processed. It follows that $E_1 = \emptyset$, which again leads to a contradiction. ◀

Now let us analyze the running time of our algorithm.

▶ **Lemma 11.** *Each insertion is processed in $O(m + n \log n)$ worst-case time.*

Proof. An insertion adds $O(n)$ edges to a single set $E_1(v)$ and causes at most a single `update` call. The running time of `update` is dominated by the time needed to find the minimum weight of a cycle going through some vertex v in some subgraph of the current graph G. By Observation 6, this time is no more than $O(m + n \log n)$. ◀

▶ **Lemma 12.** *The total time needed to process arbitrary k updates is $O\left(\sum_{i=1}^{k}(m_i + n \log n)\right)$, where m_i is the number of edges in G when the i-th update happened. In other words, the amortized update time is $O(m + n \log n)$.*

Proof. By Lemma 11, we only need to prove that the deletions take $O\left(\sum_{i=1}^{k}(m_i + n \log n)\right)$ time in total. The cost of removing the edges from the sets E_0 and $E_1(w)$, $w \in V$, can be charged to the insertions which added those edges to the graph.

After updating the edge set, a deletion is handled using a number of `update`(v) runs, in the order in which vertices v appear in Q. At most one of these runs leaves $E_1(v)$ non-empty afterwards. We charge the cost of this run to the considered deletion. For all other `update`(v) runs during that deletion, they empty the set $E_1(v)$ that previously was non-empty. As a result, we can charge the cost of that run to the last insertion of edges centered at v that happened before the considered deletion.

We need to prove two things. First of all, to see that no insertion is charged twice, note that after an insertion is charged for the first time, $E_1(v)$ is emptied. So, before `update`(v) is called next time when handling a deletion, new edges have to be added to $E_1(v)$, which can only happen during another later insertion centered at v.

We also have to prove that just before $E_1(v)$ is emptied in `update`(v), the number of edges in $G_0 + E_1(v)$ is $O(|E'|)$, where E' is the edge set of G immediately after the last insertion I centered at v happened. To this end, we prove $E(G_0) \cup E_1(v) \subseteq E'$.

We clearly had $E_1(v) \subseteq E'$ immediately after I. Afterwards no more elements were added to $E_1(v)$ (albeit some might have been removed), so we still have $E_1(v) \subseteq E'$.

Now suppose there is an edge $e \in E(G_0)$ with $e \notin E'$. Then, since $G_0 \subseteq G$, e was inserted into G after the insertion I, as a result of a later insertion I' centered at some $w \neq v$. The edge e could have been added to G_0 only if $E_1(w)$ was emptied inside `update`(w) immediately afterwards, but before `update`(v) was called. Since I was the last insertion centered at v before `update`(v) was called, both v and w were in Q when `update`(w) was called. This is a contradiction: `update` is always called on the earliest element of Q, whereas the fact that I happened before I' implies that v lied earlier than w in Q when `update`(w) was called. ◀

▶ Remark 13. When handling a deletion, we could in principle call `update`(v) for vertices v with $E_1(v) \neq \emptyset$ in arbitrary order, as opposed to in the order of least recent centered insertions. However, then one could only show a weaker total update time bound of $O(k(m_{\max} + n \log n))$, where $m_{\max}$ is the maximum number of edges in G during the first k updates.

3.1 Negative weights

In this section we extend the obtained basic algorithm to also work with negative edges. Recall that we still assume $\mu \geq 0$. Note that the case $\mu = 0$ is equivalent to dynamically maintaining whether G has a negative cycle. Recall that if G has a negative cycle, $\phi(G) = -\infty$.

Unfortunately, in presence of negative weights or cycles we cannot simply use the algorithm behind Observation 6 to find the minimum weight cycle through a vertex v in $G_0 + E_1(v)$ as we did in `update`(v). Instead, we use the following lemma.

▶ **Lemma 14.** *Let H be a digraph with no negative cycles. Let $p : V \to \mathbb{R}$ be a feasible price function of H. Let F be a set of edges centered at some vertex v.*

Then in $O(m+n\log n)$ time one can find the minimum weight of a cycle going through v in $H+F$. Moreover, if $H+F$ contains no negative cycles, within the same time bound one can produce a feasible price function on $H+F$.

Proof. Clearly, since H has no negative cycles, a negative cycle in $H+F$ has to go through v. Let E_v^+ be the set of edges in $H+F$ incoming to v. Note that $H' = H+F-E_v^+$ has no negative cycles. Moreover, since H' differs from H by edges incident to v, the edge costs reduced by p are non-negative for all edges of H' possibly except the outgoing edges of v. However, since v has no incoming edges in H', a price function p' obtained from p by sufficiently increasing $p(v)$ (e.g., to $\max\{p(u) - w(vu) : vu \in E(H')\}$) is a feasible price function of H'. With price function p' in hand, we can compute distances from v in H' using Dijkstra's algorithm in $O(m+n\log n)$ time.

Now let $x = \min_{uv\in E_v^+}\{\delta_{H'}(v,u) + w_{H'}(uv)\}$. Observe that x is indeed the minimum weight of a simple cycle in $H+F$. Moreover, $x \geq 0$ implies that $p^*(y) := \delta_{H'}(v,y) = \delta_{H+F}(v,y)$ is a feasible price function on the induced subgraph of $(H+F)[R]$ reachable from v. To extend that price function p^* on $R \subseteq V$ to entire V, it is enough to set $p^*(z) = p(z) + M$ for all $z \in V \setminus R$, where M is a sufficiently large number. To see that, note that p^* is clearly a feasible price function on $(H+F)[R]$, $(H+F)[V\setminus R]$, and there are no edges from R to $V\setminus R$ in $H+F$. For edges $zy \in E(H+F)\cap((V\setminus R)\times R)$ we have $w_{H+F}(zy) + p^*(z) - p^*(y) = w_{H+F}(zy) + p(z) + M - p^*(y)$. For

$$M = \max\{p^*(y) - p(z) - w_{H+F}(zy) : zy \in E(H+F)\cap((V\setminus R)\times R)\},$$

all the required reduced costs are non-negative. ◀

Now, given Lemma 14, we modify the basic algorithm as follows. In addition to the partition of E into E_0 and E_1, we always maintain a feasible price function p_0 on G_0. Then, in `update`(v), we use Lemma 14 to find the minimum weight x of a cycle in $G_0 + E_1(v)$. If the edges $E_1(v)$ are moved to E_0 (and thus $x \geq 0$ since $\phi(G_0+E_1(v)) \geq \mu \geq 0$), we update the price function p_0 to that produced by Lemma 14. Since the worst-case cost of running the algorithm from Lemma 14 matches that of Observation 6, the time analysis remains unchanged. Lemmas 9, 10, 12 and 14 together imply Theorem 7.

▶ Remark 15. For the problem of fully dynamically maintaining the information whether G contains a *negative* cycle (i.e., the special case $\mu = 0$) there exists a better algorithm with $O(m+n\log n)$ *worst-case* (as opposed to only amortized) update time bound (see Theorem 23). In fact, we make use of that algorithm when obtaining exact algorithms with good worst-case bounds in Section 6. The main idea is to generalize the problem to maintaining a minimum cost circulation in the graph G with imposed unit vertex/edge capacities (the details can be found in the full version). This resembles Gabow's reduction of single-source shortest paths with negative weights to the minimum cost perfect matching problem [22]. However, the min-cost circulation based algorithm is not as robust when it comes to obtaining fully dynamic algorithms for planar graphs (described in Section 5).

4 A fully dynamic $(1+\epsilon)$-approximate algorithm

In this section we show how Lemma 12 can be used to obtain an $(1+\epsilon)$-approximate minimum weight cycle algorithm, for any $\epsilon \in (0,1]$. Suppose $c \in \mathbb{R}$ ($C \in \mathbb{R}$) is a lower bound (an upper bound, respectively) on the weight of a *positive* cycle in G.

Suppose first that G has positively weighted edges. In order to convert the decision version from Section 3, all we have to do is to run it simultaneously with $\mu = (1+\epsilon)^k$ for all integers $k = \lceil \log_{1+\epsilon}(c) \rceil, \ldots, \lceil \log_{1+\epsilon}(C) \rceil$. . To maintain an approximate minimum weight of a cycle G, one only needs to keep track of the minimum k such that the fully dynamic decision algorithm for $(1+\epsilon)^k$ returns yes. If no such k exists, G is acyclic since $\phi(G) < \infty$ implies $\phi(G) \leq C$. Otherwise, we have $(1+\epsilon)^{k-1} \leq \phi(G) < (1+\epsilon)^k$, so indeed $(1+\epsilon)^k$ approximates $\phi(G)$ with multiplicative error no more than $(1+\epsilon)$. Since each of the $O(\log_{1+\epsilon}(C) - \log_{1+\epsilon}(c)) = O(\log(C/c)/\epsilon)$ decision algorithms has $O(m + n\log n)$ amortized update time, the amortized time of the approximate algorithm is $O((m+n\log n)\log(C/c)/\epsilon)$.

The same bound can be achieved even if G has non-positive edges (without, however, changing the definition of c and C) by extending each threshold data structure as described in Section 3.1. Apart from the data structures for thresholds $\mu = (1+\epsilon)^k$, we also need two more threshold cycle detection data structures: one for $\mu = 0$ to detect a negative cycle, and one for $\mu = c$ to detect whether $\phi(G) = 0$. We have thus proved Theorem 1.

▶ **Theorem 1.** *Let G be an initially empty fully dynamic real-weighted digraph such that the weight of each positive weight cycle in G always belongs to the interval $[c, C]$, $c, C \in \mathbb{R}$.*

There exists an algorithm maintaining an estimate ϕ' satisfying $\phi(G) \leq \phi' \leq (1+\epsilon)\phi(G)$ under vertex updates to G with amortized update time $O((m + n\log n) \cdot \log(C/c)/\epsilon)$.

5 Dynamic algorithms for cycles, cuts and flows in planar graphs

In this section we argue that the fully dynamic threshold cycle detection algorithm can be implemented on planar directed graphs using the known dynamic distance oracles on planar graphs. Since the reduction in Section 4 uses the threshold data structure in a black-box way, this will imply an $(1+\epsilon)$-approximate minimum weight cycle algorithm.

Using known reductions based on plane duality, this will yield fully dynamic $(1+\epsilon)$-approximate algorithms for maintaining (1) the capacity of a global min-cut in a plane digraph, (2) the value of maximum s,t-flow in a plane digraph.

The algorithms in this section handle *edge updates*, as opposed to more general vertex updates as was the case in the previous sections. Observe that achieving sublinear update time for vertex updates is not possible in general since a vertex update may need up to $\Theta(n)$ space to be described. More concretely, we will allow a single update to either insert or remove a single edge uv, provided that this update preserves planarity of G. In the cut/flow applications we will additionally need to assume that the edge insertions are *embedding preserving*, i.e., u and v lie on a single face of the current embedding of G.

Kaplan et al. [31], based on earlier work [21, 34], showed a dynamic distance oracle for real-weighted plane graphs undergoing edge weight updates. As argued in [11], their bound also holds if arbitrary, not necessarily embedding-preserving, edge updates are allowed.

▶ **Theorem 16** ([11, 21, 31, 34]). *Let G be a real-weighted planar digraph. There exists a fully dynamic algorithm supporting edge insertions and deletions in $\widetilde{O}(n^{2/3})$ worst-case time, such that for any query vertices s, t, the shortest $s \to t$ path in G can be computed in $\widetilde{O}(n^{2/3})$ time. If an edge insertion creates a negative cycle in G, the update algorithm reports it and refuses to perform that insertion. Edge insertions are not required to be embedding preserving.*

Fully dynamic threshold- and minimum weight cycles. Consider using the fully dynamic threshold cycle detection algorithm of Section 3 in the edge update scenario. Suppose that that algorithm attempts to moves edges from E_1 to E_0 single edge at a time. This does not influence correctness; the efficiency of processing a *node update* could deteriorate though (which we do not mind). Then, the *amortized* update time to process the update involving an edge uv can be actually bounded by the sum of times needed to:

1. update the set $E_1(u)$ to reflect the graph update,
2. if uv is deleted, remove uv from G_0,
3. for some $xy \in E$, find the minimum weight of a cycle going through xy in $G_0 + xy$,
4. if $\phi(G_0 + xy) \geq \mu$, insert the edge xy into G_0.

Clearly, item 1 takes constant time. If we store the (planar) graph G_0 in the data structure of Theorem 16, items 2-4 above all require $\widetilde{O}(n^{2/3})$ time. Indeed, items 2 and 4 translated to a single edge update to that data structure, whereas item 3 amounts to computing $\delta_{G_0}(y, x) + w_G(xy)$ using a single query. We thus obtain the following analogue of Theorem 7.

▶ **Theorem 17.** *Let G be a real-weighted planar digraph and let $\mu \geq 0$. There exist a fully dynamic algorithm maintaining whether $\phi(G) < \mu$ and supporting planarity-preserving edge insertions and deletions in $\widetilde{O}(n^{2/3})$ amortized time.*

Since Theorem 1 uses the threshold data structure in a black-box way, we obtain:

▶ **Theorem 18.** *Let G be a fully dynamic real-weighted planar digraph G such that the weight of any positive cycle in G always lies in the interval $[c, C]$.*

There exists an algorithm maintaining the minimum weight cycle in G under planarity preserving edge insertions and deletions with amortized update time $\widetilde{O}(n^{2/3} \log{(C/c)}/\epsilon)$.

Note that Theorem 18 immediately implies Theorem 2.

Fully dynamic directed cuts and flows. Let G be a *plane embedded* digraph with real edge capacities in $\{0\} \cup [1, W]$. Wlog. we assume that every edge e in G has its reverse e^R of capacity 0 embedded into the same curve. We can then think of any edge as traversable in both directions, but the cost of such a traversal is 0 if the edge is traversed in the reverse direction. This assumption clearly does not influence values of max-flows or min-cuts in G, but makes the dual graph G^* possess certain useful properties. We call a cycle in G^* *non-trivial* if it is not of the form ee^R for some edge $e \in E(G^*)$ and its reverse e^R.

We now state well-known properties relating flows/cuts in G to cycles in the dual G^*.

▶ **Lemma 19** (see e.g. [35]). *The global minimum cut in a plane graph G corresponds to the minimum weight non-trivial cycle in G^*.*

▶ **Lemma 20** ([19, 30, 39]). *Let G be a plane digraph with some fixed source s and sink t. For $f \geq 0$, let $G_{P,f}$ be a plane graph obtained from G adding an embedded $s \to t$ path P such that for each edge e of P, the capacity of e is f, whereas the capacity of e^R is $-f$.*

There exists an s,t-flow of value f in G if and only if the dual $G^*_{P,f}$ of $G_{P,f}$ does not contain negative cycles.

By Lemma 19, maintaining the (approximate) global min-cut dynamically under edge *embedding preserving* insertions/deletions can be reduced to maintaining the (approximate) minimum weight non-trivial cycle in the dual under vertex splits and edge contractions.

Let us now explain how such operations can be simulated using $O(1)$ updates to the data structure of Theorem 18 maintained on a certain augmented version G^*_1 of G^*, so that the minimum weights of a non-trivial cycle in G^* and G^*_1 are equal. A similar reduction has been previously described in [28, 35]. Each vertex v of the dual G^* corresponds in G^*_1 to a path P_v of $\deg_{G^*}(v)$ vertices connected using 0-weight edges traversable in both directions. For an edge $vu \in E(G^*)$ that is the i-th in (some) clockwise edge ring of v, and j-th in (some) clockwise edge ring of u, the i-th vertex of P_v is connected by an edge of weight $w_{G^*}(vu)$ with the j-th vertex of P_u. This way, (1) each vertex of G_1^* has constant degree, (2) each non-trivial cycle in G^* has a corresponding non-trivial cycle of the same weight in G_1^*, (3) no additional (with respect to G^*) non-trivial cycles are introduced in G_1^*.

It is not hard to verify that each edge contraction or vertex split in G^* can be reflected using $O(1)$ edge insertions or deletions issued to G_1^*.

Observe that the additional constraint that the minimum weight cycle is non-trivial does not introduce any difficulties: in the data structure of Theorem 17 we compute the minimum weight cycle through some edge e by issuing a distance query to a graph that *does not contain* that edge. However, since a minimum weight non-trivial cycle through e in $G_0 + e$ can traverse e in any of the two directions, we need to issue two distance queries instead of one.

We thus obtain the following theorem.

▶ **Theorem 21.** *Let G be a plane digraph with real capacities in $\{0\} \cup [1, W]$. There is an algorithm maintaining a $(1 + \epsilon)$-approximate estimate of the capacity of the global min-cut of G under* embedding preserving *edge updates with $\widetilde{O}(n^{2/3} \log W/\epsilon)$ amortized update time.*

To obtain a dynamic max s,t-flow algorithm, we use Lemma 20. We keep track of whether there exists a negative cycle (i.e., we set $\mu = 0$) in the dual of a graph a $G_{P,f}$, where $f = (1 + \epsilon)^k$, for each $k = 0, \ldots, \lceil \log_{1+\epsilon}(nW) \rceil$. Similarly as was the case for global min-cut, one can simulate the effect that an embedding preserving edge update in G has on the negative cycles of the dual of $G_{P,f}$ using $O(1)$ updates to the data structure of Theorem 17 maintained on an analogous augmentation $(G_{P,f})_1^*$ of $G_{P,f}^*$.

There is one subtle detail about how $G_{P,f}$ is updated when G is subject to embedding preserving edge insertions and deletions. Note that Lemma 20 requires us to embed any additional simple $s \to t$ path P into G. Embedding P into G subdivides some of the original faces of G. As a result, an edge uv to be inserted inside some face F of G may cross some edges of the currently used path P in $G_{P,f}$. We deal with this problem as follows. We maintain an additional invariant that (the embedding of) the simple path P crosses each face of G at most once.

Now, when a new edge uv is inserted inside F, and P has an edge $e = xy$ inside F that would cross uv, we first remove e from $G_{P,f}$ to allow the insertion of uv. This insertion splits F into two faces F_1, F_2 such that x lies on F_1 and y lies on F_2. We now reconnect the path P by embedding two edges xu, uy with appropriate capacities as required by Lemma 20.

On the other hand, when an edge uv is removed, two faces F_1 of F_2 of G are merged into a single face F. If at most one of them F_1, F_2 contained an edge of P, we do not have to do anything. Otherwise, suppose wlog. that F_1 contains an edge $xy = e_1 \in P$, and F_2 contains an edge $ab = e_2 \in P$, such that e_1 appears before e_2 on P. Then, we remove e_1, e_2, and all edges between e_1 and e_2 on P from $G_{P,f}$, and replace them with a single edge xb embedded in F. Afterwards, the invariant is satisfied and P remains a simple path.

Finally, observe that each update to G adds $O(1)$ new edges to P in the *worst case*. An edge deletion may remove a superconstant number of edges from P, but these removals can be charged to the corresponding additions of new edges to P. To conclude, an edge update to G translates to $O(1)$ amortized edge updates to $G_{P,f}$, and as a result, to $O(1)$ amortized operations on the data structure of Theorem 17 run on the augmented dual $(G_{P,f})_1^*$. We have thus proved:

▶ **Theorem 3.** *Let G be a plane embedded digraph with real edge capacities in $\{0\} \cup [1, W]$ and a* fixed *source/sink pair s, t. There exists an algorithm maintaining a $(1 - \epsilon)$-approximate estimate of the value of maximum s,t-flow in G under* embedding preserving *edge insertions and deletions with $\widetilde{O}(n^{2/3} \log(W)/\epsilon)$ amortized update time.*

6 Exact fully dynamic algorithm for minimum weight cycle

In this section we argue that using a variant of the fully dynamic APSP algorithm of Abraham et al. [2] one can achieve subquadratic update bounds for dynamic minimum weight cycle.

We will in fact first solve a slightly more general problem that we call the *fully dynamic multiple-pairs shortest paths* (fully dynamic MPSP for short). Our goal is to have a data structure that maintains distances $\delta_G(s_i, t_i)$ for some *fixed* (throughout the course of the algorithm) k source-target pairs $(s_1, t_1), \dots, (s_k, t_k)$ subject to fully dynamic vertex updates. Obviously, the classical fully dynamic APSP corresponds to the case $k = n^2$.

In the following we sketch the approach of [2] to fully dynamic APSP. The presentation is however directed towards our goal of obtaining an MPSP data structure. Some details and proofs can be found only in [2]; we focus on the details of our adjustments.

Reduction to batch-deletion MPSP data structure. The first step is to reduce the fully dynamic problem to a certain decremental problem, called the *batch-deletion MPSP*. In this problem, we want to preprocess the input digraph G, so that one can efficiently compute MPSP in $G \setminus D$ for a subset $D \subseteq V$ that constitutes the query parameter. We assume that if $G \setminus D$ has a negative cycle, the data structure has to report its existence instead.

▶ **Lemma 22.** *Suppose we have a batch-deletion MPSP data structure with preprocessing time $T_{\text{pre}}(n, m, k)$ and worst-case query time $T_{\text{q}}(n, m, k, d)$, where $d = |D|$ is the size of the removed vertex set. Then, for any integer $\Delta > 0$, there exists a fully dynamic MPSP algorithm with worst-case update time $O(T_{\text{pre}}(n, m, k)/\Delta + T_{\text{q}}(n, m, k, \Delta) + \Delta(m + k + n \log n))$.*

Proof sketch. To obtain an amortized (as opposed to worst-case) bound from the statement, we split the timeline into phases of Δ updates. When a new phase starts, we rebuild the batch-deletion data structure from scratch on the graph G_0 at the start of the phase; this clearly incurs $O(T_{\text{pre}}(n, m_0, k)/\Delta)$ amortized time cost per update, where $m_0 = |E(G_0)|$. At some point of a phase, let $D \subseteq V$, $|D| \leq \Delta$, be the vertices touched by updates in this phase. To compute MPSP at that point, we first compute MPSP in $G_0 \setminus D = G \setminus D$ in $O(T_{\text{q}}(n, m_0, k, |D|)) = O(T_{\text{q}}(n, m_0, k, \Delta))$ time. To obtain MPSP in G, we need to check if paths going through D in G improve upon those in $G \setminus D$, i.e., we compute MPSP in G according to the equation $\delta_G(s_i, t_i) = \min\left(\delta_{G_0 \setminus D}(s_i, t_i), \min_{v \in D}\{\delta_G(s_i, v) + \delta_G(v, t_i)\}\right)$.

Observe that all distances of the form $\delta_G(\cdot, v)$ or $\delta_G(v, \cdot)$ for $v \in D$ can be obtained by running Dijkstra's algorithm to/from each such v, in $O(\Delta(m + n \log n))$ total time, as long as a feasible price function of G is given. A feasible price function can be maintained in $O(m + n \log n)$ worst-case time after a vertex update using the following theorem, whose proof is deferred to the full version.

▶ **Theorem 23.** *Let G be an initially empty real-weighted digraph. There exists an algorithm maintaining the information whether G has a negative cycle and supporting vertex updates in $O(m + n \log n)$ worst-case time. Additionally, whenever $\phi(G) \geq 0$, the algorithm maintains a feasible price function p of G.*

Theorem 23 is also used to keep track of whether the current G has a negative cycle. Once these distances are available, the distances $\delta_G(s_i, t_i)$ can be computed in $O(\Delta \cdot k)$ time.

Unfortunately, the above argument is fully valid only if either the number of edges m is of the same order throughout, i.e., $m_0 = O(m)$, or it cannot drop by more than a constant factor during a single phase, e.g., $m = \Omega(n\Delta)$. If, say, $T_{\text{pre}}(n, m, k) = \Theta(nm)$, $\Delta = n^{1/3}$ and $m = n^{5/4} = m_0$ at the beginning of the phase, and during $n^{1/4}$ first updates in that phase m

gets decreased to $O(n)$, then the total update cost coming from the preprocessing in this phase is $\Theta(nm_0) = \Theta(n^{9/4})$. If the amortized update time coming from the preprocessing was indeed $O(T_{\text{pre}}(n, m, k)/\Delta)$, the the total update cost coming from these terms in that phase would be $O(n^{1/4} \cdot nm_0/\Delta + \Delta \cdot n^2/\Delta) = O(n^{13/6})$, i.e., polynomially less.

We circumvent this problem[4] as follows. We build the batch-deletion MPSP data structure on the graph $G_0' = G_0 \setminus D^*$ instead of G_0, where D^* is the set of Δ vertices of G_0 with highest degree. Then, Dijkstra's algorithm is used to separately compute shortest paths through $D \cup D^*$ in G, as opposed to only through D. Clearly, the cost of such computation remains $O(\Delta(m + k + n \log n))$. However, the update cost coming from the batch-deletion MPSP data structure is decreased to $O(T_{\text{pre}}(n, m_0', k)/\Delta + T_{\text{q}}(n, m_0', k, \Delta))$, where m_0' is the number of edges in G_0'. It is hence enough to observe that $m_0' \leq m$ throughout this phase. Indeed, the updates centered at vertices D cannot remove more than $\sum_{v \in D} \deg_{G_0}(v)$ edges out of those originally contained in G_0. As a result, $m \geq m_0 - \sum_{v \in D} \deg_{G_0}(v)$. On the other hand, by removing D^* from G_0 we remove at least $\frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v)$ edges from G_0, i.e, $m_0' \leq m_0 - \frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v)$. We obtain $m_0' \leq m$ as follows:

$$m_0' \leq m_0 - \frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v) \leq m_0 - \frac{1}{2} \sum_{v \in D} \deg_{G_0}(v) \leq m_0 + \frac{1}{2}(m - m_0) = \frac{1}{2} m_0' + \frac{1}{2} m.$$

Since the amortization comes only from a (costly) rebuilding step after every Δ updates, turning the amortized bound into a worst-case one is standard, see e.g., [2, Section 2]. ◀

The batch-deletion data structure. Abraham et al [2] showed a batch-deletion APSP data structure with $\widetilde{O}(n^3)$ preprocessing time and $\widetilde{O}(n^2\sqrt{nd})$ query time which, by Lemma 22, implies $\widetilde{O}(n^{2+2/3})$ worst-case update time for fully dynamic APSP. Their batch-deletion data structure is Monte Carlo randomized and produces answers correct with high probability. We generalize this data structure to MPSP and non-dense graphs.

▶ **Theorem 24.** *There exists a Monte Carlo randomized batch-deletion MPSP data structure with $O((m + k)n \log^2 n)$ preprocessing and $O((m + n \log n + k)\sqrt{nd} \log n)$ query time. The answers produced are correct with high probability.*

Before we prove Theorem 24, let us show how it can be used to obtain fully dynamic MPSP and minimum weight cycle algorithms.

By choosing $\Delta = n^{1/3} \log^{2/3} n$, and applying Lemma 22, we obtain:

▶ **Theorem 5.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic MPSP data structure supporting vertex updates with $O((m+n \log n+k)n^{2/3} \log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

Now consider the fully dynamic minimum weight cycle problem. The minimum weight of a cycle in G is given by $\phi(G) = \min_{uv \in E(G)}\{\delta_G(v, u) + w_G(uv)\}$. As a result, after each update it is enough to recompute distances $\delta_G(s_l, t_l)$ in G for $k = m$ pairs (s_l, t_l) such that $t_l s_l \in E(G)$. If the edge set of G was fixed (and, for example, the updates were only allowed to change edge weights), so would be the set of source-target pairs of our interest. Hence, we could simply use the fully dynamic MPSP data structure of Theorem 5 in a black-box way. However, in general, $E(G)$ is not fixed and we need to be more careful.

[4] This problem does not arise in [2], since there m is assumed to be $\Theta(n^2)$ throughout.

We proceed as follows. In the reduction of Lemma 22, we will always build a batch-deletion MPSP data structure with the set of source-target pairs equal to the edge set used to build that data structure reversed. This means that at any point of the phase, we can compute the minimum weight cycle in $G \setminus (D^* \cup D) = G_0 \setminus (D^* \cup D)$ in $O(T_{\text{pre}}(n,m,m)/\Delta + T_{\text{q}}(n,m,m,\Delta)) = O((m+n\log n)n^{2/3}\log^{4/3} n)$ worst-case time. Since $G_0 \setminus (D^* \cup D)$ contains only a subset of edges of G_0, reading the subset of entries of the distance matrix of $G_0 \setminus (D^* \cup D)$ corresponding to reversed edges of $E(G_0)$ is enough to this end. In order to find the minimum weight cycle going through some vertex of $D^* \cup D$ in G, we just run the algorithm of Observation 6 (or, more generally, in presence of negative edges – the algorithm of Lemma 14 with a feasible price function maintained by the algorithm of Theorem 23) $|D^* \cup D| = O(\Delta)$ times. This costs $O(\Delta(m + n\log n)) = \widetilde{O}(mn^{1/3})$ time.

▶ **Theorem 4.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic algorithm maintaining $\phi(G)$ under vertex updates with $O((m+n\log n)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

6.1 Overview of the batch-deletion MPSP data structure

Let us now sketch the idea behind our generalization of the batch-deletion data structure of [2]. Due to space constraints, the detailed description can be found in the full version.

We first need to refer to some details of the construction of Abraham et al. [2]. The batch-deletion data structure separately handles recomputing shortest paths of hop-length at least $\sqrt{n/d}$ ("long" paths), and separately "short" shortest paths – with hop-lengths in the intervals of the form $[h/2, h)$ for $O(\log n)$ values $h = 2^1, 2^2, \ldots, \sqrt{n/d}$.

The main difficulty lies in handling short paths, whereas handling long paths is an easier task. The key idea (which dates back to Thorup [46]) is to compute an ordered subset $\{v_1, \ldots, v_\ell\} \subseteq V$ with the following properties. Let $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. Let $\mathcal{P}_i$ be the set of shortest $\leq h$-hop paths from/to v_i in G_i. Then:

(1) For any $s, t \in V$, an $s \to t$ path not longer than the shortest $\leq h$-hop $s \to t$ path in G can be obtained by stitching, for some $i \in \{1, \ldots, \ell\}$, the $s \to v_i$ and $v_i \to t$ paths of $\mathcal{P}_i$.
(2) For any $x \in V$, x lies on at most $\widetilde{O}(hn)$ paths from $\bigcup_{i=1}^{\ell} \mathcal{P}_i$.

Such an ordering, along with the paths $\mathcal{P}_i$, can be computed in $\widetilde{O}(nmh)$ time deterministically (then we have $\ell = n$), or in $\widetilde{O}(nm)$ time using randomization (then $\ell = \widetilde{O}(n/h)$). Each subsequent vertex v_i in the ordering is picked to be, roughly speaking, the "most congested" one out of $V \setminus \{v_1, \ldots, v_{i-1}\}$, i.e., the one that has not been picked yet and appears most often on the previously constructed paths $\bigcup_{j=1}^{i-1} \mathcal{P}_j$.

Given the above, Abraham et al. [2] show that after removing any $D \subseteq V$ from G, the "short" paths in G can be recomputed by:

(1) constructing a number of *sketch graphs* $H_1, \ldots, H_\ell$, where $H_i \subseteq G_i \setminus D$,
(2) rebuilding destroyed (by the removal of D) paths from $\mathcal{P}_i$ by running Dijkstra's algorithm from/to v_i on H_i,
(3) stitching the reconstructed paths back to obtain paths at leas as good as the actual shortest $\leq h$-hop paths in G.

Abraham et al. [2] prove that if we denote by U_i the set of vertices u such that either of the paths $u \to v_i$ or $v_i \to u$ from $\mathcal{P}_i$ has been destroyed by removing D, $d = |D|$, then we have $\sum_{i=1}^{\ell} |U_i| = \widetilde{O}(hnd)$, and the total number of edges M in the sketch graphs is

$$M = O\left(\sum_{i=1}^{\ell}\left(n + \sum_{u \in U_i} \deg_G(u)\right)\right).$$

It is easy to see that $M = \Omega(n\ell)$, and $M = \tilde{O}(hn^2 d)$. Moreover, for each rebuilt path $u \to v_i$ or $v_i \to u$, stitching takes additional $\Theta(n)$ time – as one needs to traverse through $\Theta(n)$ source-target pairs that might benefit from this – for a total of $\widetilde{O}(hn^2 d)$ time. Since h ranges from $O(1)$ to $\Theta\left(\sqrt{n/d}\right)$, rebuilding short paths takes $\Omega(n^2)$ and $\widetilde{O}(n^2\sqrt{nd})$ time as claimed.

Now, to obtain our improved $\widetilde{O}((m+k)\sqrt{nd})$ bound on batch deletion for sparse graphs and small number k of source-target paths (s_i, t_i) of interest, we make two main adjustments.

First of all, we show that even smaller sketch graphs H_i – with $O\left(\sum_i^{\ell}\sum_{u\in U_i}\deg_G(u)\right)$ edges in total – can be used, thus eliminating the $\Omega(n\ell)$ term, which for small h is $\Omega(n^2)$.

More importantly, we use a different *weighted* scheme for picking the ordered subset $\{v_1, \ldots, v_\ell\}$. Let us denote by K the undirected graph on V whose edges correspond to the source-target pairs (s_i, t_i) of interest. In our scheme, the congestion that a previously computed $\leq h$-hop path $P = v_i \to u$ (or $P = u \to v_i$) incurs upon some vertex x with $x \in V(P)$ is $\deg_G(u) + \log n + \deg_K(u)$, as opposed to 1 in [2]. This makes the total congestion of each vertex x in the process possibly increase to $\tilde{\Theta}(h(m + n\log n + k))$, as opposed to $\widetilde{O}(hn)$ in [2]. However, we show that the total cost of running Dijkstra's algorithm on our (more compact) sketch graphs $H_1, \ldots, H_\ell$ can be charged to the part of the total congestion of removed vertices D coming from the $[\deg_G(u) + \log n]$ terms, which is $\widetilde{O}(dh(m + n\log n))$. A similar argument applies to the cost of restitching, which we prove to be $\widetilde{O}(dhk)$.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 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.

3 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.13.

4 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12(4):615–638, 1991. doi:10.1016/0196-6774(91)90036-X.

5 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.

6 Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.16.

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

8 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1355–1365. SIAM, 2011. doi:10.1137/1.9781611973082.104.

9 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.

10 Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 943–954. ACM, 2019. doi:10.1145/3313276.3316373.

11 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 31:1–31:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.31.

12 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.

13 Shiri Chechik and Gur Lifshitz. Optimal girth approximation for dense directed graphs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 290–300. SIAM, 2021. doi:10.1137/1.9781611976465.19.

14 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1010–1023. ACM, 2020. doi:10.1145/3357713.3384330.

15 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2478–2496. SIAM, 2021. doi:10.1137/1.9781611976465.147.

16 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28:1–28:30, 2015. doi:10.1145/2736283.

17 Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.35.

18 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.

19 Jeff Erickson. Maximum flows and parametric shortest paths in planar graphs. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804. SIAM, 2010. doi:10.1137/1.9781611973075.65.

20 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. arXiv:2010.00937.

21 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.

22 Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985. doi:10.1016/0022-0000(85)90039-X.

23 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.

24 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.

25 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.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

27 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. doi:10.1137/0207033.

28 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.

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

30 Donald B. Johnson. Parallel algorithms for minimum cuts and maximum flows in planar networks. *J. ACM*, 34(4):950–967, 1987. doi:10.1145/31846.31849.

31 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.

32 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.

33 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.

34 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.

35 Hung-Chun Liang and Hsueh-I Lu. Minimum cuts and shortest cycles in directed planar graphs via noncrossing shortest paths. *SIAM J. Discret. Math.*, 31(1):454–476, 2017. doi:10.1137/16M1057152.

36 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018. doi:10.1137/1.9781611975031.80.

37 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in o(n loglogn) time. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2011. doi:10.1007/978-3-642-23719-5_14.

38 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum cut of directed planar graphs in $O(n \log \log n)$ time. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 477–494. SIAM, 2018. doi:10.1137/1.9781611975031.32.

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

40 James B. Orlin and Antonio Sedeño-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1866–1879. SIAM, 2017. doi:10.1137/1.9781611974782.122.

41 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. doi:10.1016/S0304-3975(03)00402-X.

42 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.27.

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

44 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. doi:10.1137/090776573.

45 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.

46 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.

47 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

48 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.