Min-Cost Flow in Unit-Capacity Planar Graphs

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

Piotr Sankowski
Institute of Informatics, University of Warsaw, Poland
sank@mimuw.edu.pl

Abstract

In this paper we give an $\widetilde{O}((nm)^{2/3}\log C)$ time algorithm for computing min-cost flow (or min-cost circulation) in unit capacity planar multigraphs where edge costs are integers bounded by C. For planar multigraphs, this improves upon the best known algorithms for general graphs: the $\widetilde{O}(m^{10/7}\log C)$ time algorithm of Cohen et al. [SODA 2017], the $O(m^{3/2}\log(nC))$ time algorithm of Gabow and Tarjan [SIAM J. Comput. 1989] and the $\widetilde{O}(\sqrt{n}m\log C)$ time algorithm of Lee and Sidford [FOCS 2014]. In particular, our result constitutes the first known fully combinatorial algorithm that breaks the $\Omega(m^{3/2})$ time barrier for min-cost flow problem in planar graphs.

To obtain our result we first give a very simple successive shortest paths based scaling algorithm for unit-capacity min-cost flow problem that does not explicitly operate on dual variables. This algorithm also runs in $\widetilde{O}(m^{3/2}\log C)$ time for general graphs, and, to the best of our knowledge, it has not been described before. We subsequently show how to implement this algorithm faster on planar graphs using well-established tools: r-divisions and efficient algorithms for computing (shortest) paths in so-called dense distance graphs.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases minimum-cost flow, minimum-cost circulation, planar graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.66

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.02274`.

Funding *Adam Karczmarz*: Supported by ERC Consolidator Grant 772346 TUgbOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.
Piotr Sankowski: Supported by ERC Consolidator Grant 772346 TUgbOAT.

1 Introduction

The min-cost flow is the core combinatorial optimization problem that now has been studied for over 60 years, starting with the work of Ford and Fulkerson [14]. Classical combinatorial algorithms for this problem have been developed in the 80s. Goldberg and Tarjan [18] showed an $\widetilde{O}(nm\log C)$ time weakly-polynomial algorithm for the case when edge costs are integral, and where C is the maximum edge cost. Orlin [31] showed the best-known strongly polynomial time algorithm running in $\widetilde{O}(m^2)$ time. Faster weakly-polynomial algorithms have been developed in this century using interior-point methods: Daitch and Spielman [8] gave an $\widetilde{O}(m^{3/2}\log(U+C))$ algorithm, and later Lee and Sidford [28] obtained an $\widetilde{O}(\sqrt{n}m\log(U+C))$ algorithm, where U is the maximum (integral) edge capacity.

Much attention has been devoted to the unit-capacity case of the min-cost flow problem. Gabow and Tarjan [15] gave a $O(m^{3/2}\log(nC))$ time algorithm. Lee and Sidford [28] matched this bound up to polylogarithmic factors for $m=\widetilde{O}(n)$, and improved upon it for larger densities, even though their algorithm solves the case of arbitrary integral capacities. Gabow and Tarjan's result remained the best known bound for more than 28 years – the problem

witnessed an important progress only very recently. In 2017 an algorithm that breaks the $\Omega(m^{3/2})$ time barrier for min-cost flow problem was given by Cohen et al. [7]. This algorithm runs in $\widetilde{O}(m^{10/7} \log C)$ time and is also based on interior-point methods.

It is worth noting that currently the algorithms of [7, 28] constitute the most efficient solutions for the entire range of possible densities (up to polylogarithmic factors) and are also the best-known algorithms for important special cases, e.g., planar graphs or minor-free graphs. Both of these solutions are based on interior point methods and do not shed light on the combinatorial structure of the problem.

In this paper we study the unit-capacity min-cost flow in planar multigraphs. We improve upon [7, 28] by giving the first known $\widetilde{O}((mn)^{2/3} \log C) = \widetilde{O}(m^{4/3} \log C)$ time algorithm for computing min-cost s,t-flow and min-cost circulation in planar multigraphs.[1] Our algorithm is fully combinatorial and uses the scaling approach of Goldberg and Tarjan [18]. At each scale it implements the classical shortest augmenting path approach similar to the one known from the well-known Hopcroft-Karp algorithm for maximum bipartite matching [19].

Related work. Due to immense number of works on flows and min-cost flows we will not review all of them. Instead we concentrate only on the ones that are relevant to the sparse and planar graph case, as that is the regime where our results are of importance. As already noted above the fastest algorithms for min-cost flows in planar multigraphs are implied by the algorithms for general case. This, however, is not the case for maximum flow problem. Here, the fastest algorithms are based on planar graph duality and reduce the problem to shortest path computations. The undirected s,t-flow problem can be solved in $O(n \log \log n)$ time [22], whereas the directed s,t-flow problem can be solved in $O(n \log n)$ time [3, 11]. Even for the case with multiple source and sinks, a nearly-linear time algorithm is known [4].

These results naturally raise as an open question whether similar nearly-linear bounds could be possible for min-cost flow. Until very recently there has been no progress towards answering this open question. Partial progress was made by devising $\widetilde{O}(n^{4/3} \log C)$ time [1] and $\widetilde{O}(n^{6/5} \log C)$ time [27] algorithms for min-cost perfect matchings in bipartite planar graphs. Lahn and Raghvendra also give an $\widetilde{O}(n^{7/5} \log C)$ time minimum cost perfect matching algorithm for minor-free graphs. These algorithms can be seen as specialized versions of the Gabow-Tarjan's algorithm for the assignment problem [15].

Gabow and Tarjan [15] reduced min-cost flow problem to so-called min-cost perfect degree-constrained subgraph problem on a bipartite multigraph, which they solved by extending their algorithm for minimum cost perfect matching. Hence it seems plausible that the recent algorithm of Lahn and Raghvendra [27] can be extended to solve min-cost flow, since their algorithm builds upon the Gabow-Tarjan algorithm. The reduction presented by Gabow and Tarjan *is not* planarity preserving, though. Nevertheless, min-cost perfect matching problem can be reduced to min-cost flow problem in an efficient and planarity preserving way [29]. The opposite reduction can be done in planarity preserving way as recently shown [33]. However, this reduction is not efficient and produces a graph of quadratic size. Hence, we cannot really take advantage of it.

Overview and comparison to [1, 27]. We concentrate on the *min-cost circulation* problem, which is basically the min-cost flow problem with all vertex demands equal to 0. It is well-known [17] that the min-cost s,t-flow problem can be solved by first computing some s,t-flow

[1] It is known that simple planar graphs have $O(n)$ edges. However, multiple parallel edges (with possibly different costs) are useful in the unit-capacity min-cost flow problem, as they allow us to encode larger edge capacities. Therefore, in this paper we work with planar multigraphs.

f of requested value (e.g., the maximum value), and then finding a min-cost circulation on the residual network G_f. This reduction is clearly planarity-preserving. Since an s,t-flow of any given value (in particular, the maximum value) can be found in a planar graph in nearly-linear time (see [11]), this reduction works in nearly-linear time as well.

Our min-cost circulation algorithm resembles the recent works on minimum cost planar perfect matching [1, 27], in the sense that we simulate some already-good scaling algorithm for general graphs, but implement it more efficiently using the known and well-established tools from the area of planar graph algorithms. However, instead of simulating an existing unit-capacity min-cost flow algorithm, e.g., [15, 17], we use a very simple successive-shortest paths based algorithm that, to the best our knowledge, has not been described before.

Our algorithm builds upon the cost-scaling framework of Goldberg and Tarjan [18], similarly as the recent simple unit-capacity min-cost flow algorithms of Goldberg et al. [17]. In this framework, a notion of ϵ-optimality of a flow is used. A flow f is ϵ-optimal with respect to a price function p if for any edge $uv = e \in E(G_f)$ we have $c(e) - p(u) + p(v) \geq -\epsilon$.

Roughly speaking, the parameter ϵ measures the quality of a circulation: any circulation is trivially C-optimal wrt. p, whereas any $\frac{1}{n}$-feasible (wrt. p) circulation is guaranteed to be optimal. The general scheme is to start with a C-optimal circulation, run $O(\log(nC))$ *scales* that improve the quality of a circulation by a factor of 2, and this way obtain the optimal solution.

We show that a single scale can be solved by repeatedly sending flow along a cheapest $s \to t$ path in a certain graph G''_f with a single source s and a single sink t, that approximates the residual graph G_f. Moreover, if we send flow simultaneously along a maximal set of cheapest $s \to t$ paths at once, like in [12, 19], we finish after $O(\sqrt{m})$ augmentations. However, as opposed to [12, 19], our graph G''_f is weighted and might have negative edges. We overcome this difficulty as in the classical successive shortest path approach for min-cost flow, by using distances from the previous flow augmentation as a feasible price function that can speed-up next shortest path computation. Our algorithm also retains a nice property[2] of the Even-Tarjan algorithm that the total length (in terms of the number of edges) of all the used augmenting paths is $O(m \log m)$.

The crucial difference between our per-scale procedure and those of [15, 17] is that we do not "adjust" dual variables $p(v)$ at all while the procedure runs: we only use them to compute G''_f, and recompute them from scratch in nearly-linear time when the procedure finishes. In particular, the recent results of [1, 27] are quite complicated since, in order to simulate the Gabow-Tarjan algorithm [15], they impose and maintain additional invariants about the duals.

The only bottlenecks of our per-scale procedure are (1) shortest paths computation, (2) picking a maximal set of edge-disjoint $s \to t$ paths in an unweighted graph[3].

We implement these on a planar network using standard methods. Let $r \in [1, n]$ be some parameter. We construct a *dense distance graph* H''_f (e.g., [13, 16]) built upon an r-division (e.g., [26]) of G''_f. The graph H''_f is a compressed representation of the distances in G''_f with $O(n/\sqrt{r})$ vertices and $O(m)$ edges. Moreover, it can be updated in $\widetilde{O}(r)$ time per edge used by the flow. Hence, the total time spent on updating H''_f is $\widetilde{O}(mr)$. As we show, running our per-scale procedure on H''_f is sufficient to simulate it on G''_f. Computing distances in a dense distance graph requires $\widetilde{O}(n/\sqrt{r})$ time [13, 16]. To complete the construction, we show how

[2] Gabow-Tarjan algorithm for min-cost bipartite matching has a similar property, which was instrumental for obtaining the recent results on minimum-cost planar bipartite matching [1, 27].

[3] This is sometimes called *the blocking flow* problem and can be solved for unit capacities in linear time.

to find a maximal set of edge-disjoint paths in $\widetilde{O}(n/\sqrt{r})$ amortized time. To this end, we also exploit the properties of reachability in a dense distance graph, used previously in dynamic reachability algorithms for planar digraphs [21, 24]. This way, we obtain $\widetilde{O}(\sqrt{mn}/\sqrt{r} + mr)$ running time per scale. This is minimized roughly when $r = n^{2/3}/m^{1/3}$.

Recall that Lahn and Raghvendra [27] obtained a polynomially better (than ours) bound of $\widetilde{O}(n^{6/5} \log C)$, but only for planar min-cost perfect matching problem. To achieve that, they use an additional idea due to Asathulla et al. [1]. Namely, they observe that by introducing vertex weights, one can make augmenting paths avoid edges incident to boundary vertices, thus making the total number of pieces "affected" by augmenting paths truly-sublinear in n. It is not clear how to apply this idea to the min-cost flow problem without making additional assumptions about the structure of the instance, like bounded-degree (then, there are only $O(n/\sqrt{r})$ edges incident to boundary vertices of an r-division), or bounded vertex capacities (so that only $O(1)$ units of flow can go through each vertex; this is satisfied in the perfect matching case). This phenomenon seems not very surprising once we recall that such assumptions lead to better bounds even for general graphs: the best known combinatorial algorithms for min-cost perfect matching run in $O(n^{1/2}m \log(nC))$ time, whereas for min-cost flow in $O(m^{3/2} \log(nC))$ time [15, 17].

Organization of the paper. In Section 2 we introduce the notation and describe the scaling framework of [18]. Next, in Section 3, we describe the per-scale procedure of unit-capacity min-cost flow for general graphs. Finally, in Section 4 we give our algorithm for planar graphs. Due to limited space, some of the missing proofs can only be found in the full version of the paper.

2 Preliminaries

Let $G_0 = (V, E_0)$ be the input directed multigraph. Let $n = |V|$ and $m = |E_0|$. Define $G = (V, E)$ to be a multigraph such that $E = E_0 \cup E_0^{\mathrm{R}}$, $E_0 \cap E_0^{\mathrm{R}} = \emptyset$, where E_0^{R} is the set of *reverse edges*. For any $uv = e \in E$, there is an edge $e^{\mathrm{R}} \in E$ such that $e^{\mathrm{R}} = vu$ and $(e^{\mathrm{R}})^{\mathrm{R}} = e$. We have $e \in E_0$ iff $e^{\mathrm{R}} \in E_0^{\mathrm{R}}$.

Let $u : E_0 \to \mathbb{R}_+$ be a *capacity function*. A *flow* is a function $f : E \to \mathbb{R}$ such that for any $e \in E$ $f(e) = -f(e^{\mathrm{R}})$ and for each $e \in E_0$, $0 \le f(e) \le u(e)$. These conditions imply that for $e \in E_0$, $-u(e) \le f(e^{\mathrm{R}}) \le 0$. We extend the function u to E by setting $u(e^{\mathrm{R}}) = 0$ for all $e \in E_0$. Then, for all edges $e \in E$ we have $-u(e^{\mathrm{R}}) \le f(e) \le u(e)$. The *unit capacity* function satisfies $u(e) = 1$ for all $e \in E_0$.

The *excess* $\mathrm{exc}_f(v)$ of a vertex $v \in V$ is defined as $\sum_{uv=e\in E} f(e)$. Due to anti-symmetry of f, $\mathrm{exc}_f(v)$ is equal to the amount of flow going into v by the edges of E_0 minus the amount of flow going out of v by the edges of E_0. The vertex $v \in V$ is called an *excess* vertex if $\mathrm{exc}_f(v) > 0$ and *deficit* if $\mathrm{exc}_f(v) < 0$. Let X be the set of excess vertices of G and let D be the set of deficit vertices. Define the *total excess* Ψ_f as the sum of excesses of the excess vertices, i.e., $\Psi_f = \sum_{v\in X} \mathrm{exc}_f(v) = \sum_{v\in D} -\mathrm{exc}_f(v)$.

A flow f is called a *circulation* if there are no excess vertices, or equivalently, $\Psi_f = 0$.

Let $c : E_0 \to \mathbb{Z}$ be the input *cost* function. We extend c to E by setting $c(e^{\mathrm{R}}) = -c(e)$ for all $e \in E_0$. The *cost* $c(f)$ of a flow f is defined as $\frac{1}{2}\sum_{e\in E} f(e)c(e) = \sum_{e\in E_0} f(e)c(e)$.

To send a unit of flow through $e \in E$ means to increase $f(e)$ by 1 and simultaneously decrease $f(e^{\mathrm{R}})$ by 1. By sending a unit of flow through e we increase the cost of flow by $c(e)$. *To send a unit of flow through a path* P means to send a unit of flow through each edge of P. In this case we also say that we *augment flow* f *along path* P.

The *residual network* G_f of f is defined as (V, E_f), where $E_f = \{e \in E : f(e) < u(e)\}$.

Price functions and distances. We call any function $p : V \to \mathbb{R}$ a *price function* on G. The *reduced cost* of an edge $uv = e \in E$ wrt. p is defined as $c_p(e) := c(e) - p(u) + p(v)$. We call p a *feasible price function* of G if each edge $e \in E$ has nonnegative reduced cost wrt. p.

It is known that G has no negative-cost cycles (negative cycles, in short) if and only if some feasible price function p for G exists. If G has no negative cycles, distances in G (where we interpret c as a *length* function) are well-defined. For $u, v \in V$, we denote by $\delta_G(u, v)$ the distance between u and v, or, in other words, the length of a shortest $u \to v$ path in G.

▶ **Fact 1.** *Suppose G has no negative cycles. Let $t \in V$ be reachable in G from all vertices $v \in V$. Then the* distance to *function $\delta_{G,t}(v) := \delta_G(v, t)$ is a feasible price function of G.*

For $A, B \subseteq V(G)$ we sometimes write $\delta_G(A, B)$ to denote $\min\{\delta_G(u, v) : u \in A, v \in B\}$.

Planar graph toolbox. An *r-division* of a simple undirected plane graph G is a collection of $O(n/r)$ edge-induced subgraphs of G, called *pieces*, whose union is G and such that each piece P has $O(r)$ vertices and $O(\sqrt{r})$ *boundary vertices*. The boundary vertices ∂P of a piece P are the vertices of P shared with some other piece.

An *r-division with few holes* has an additional property that for each piece P, (1) P is connected, (2) there exist $O(1)$ faces of P whose union of vertex sets contains ∂P.

Let $G_1, \ldots, G_\lambda$ be some collection of plane graphs, where each G_i has a distinguished boundary set ∂G_i lying on $O(1)$ faces of G_i. A *distance clique* $\mathrm{DC}(G_i)$ of G_i is defined as a complete digraph on ∂G_i such that the cost of the edge uv in $\mathrm{DC}(G_i)$ is equal to $\delta_{G_i}(u, v)$.

▶ **Theorem 2** (MSSP [6, 25])**.** *Suppose a feasible price function on G_i is given. Then the distance clique $DC(G_i)$ can be computed in $O((|V(G_i)| + |E(G_i)| + |\partial G_i|^2) \log |V(G_i)|))$ time.*

The graph $\mathrm{DDG} = \mathrm{DC}(G_1) \cup \ldots \cup \mathrm{DC}(G_\lambda)$ is called a *dense distance graph*[4].

▶ **Theorem 3** (FR-Dijkstra [13, 16])**.** *Given a feasible price function of DDG, single-source shortest paths in DDG can be computed in $O\left(\sum_{i=1}^{\lambda} |\partial G_i| \frac{\log^2 n}{\log^2 \log n}\right)$ time, where $n = |V(DDG)|$.*

Scaling framework for minimum-cost circulation. The following fact characterizes minimum circulations.

▶ **Fact 4** ([32])**.** *Let f be a circulation. Then $c(f)$ is minimum iff G_f has no negative cycles.*

It follows that a circulation f is minimum if there exists a feasible price function of G_f.

▶ **Definition 5** ([2, 18, 34])**.** *A flow f is ϵ-optimal wrt. price function p if for any $uv = e \in E_f$, $c(e) - p(u) + p(v) \geq -\epsilon$.*

The above notion of ϵ-optimality allows us, in a sense, to measure the optimality of a circulation: the smaller ϵ, the closer to the optimum a circulation f is. Moreover, if we deal with integral costs, $\frac{1}{n+1}$-optimality is equivalent to optimality.

▶ **Lemma 6** ([2, 18])**.** *Suppose the cost function has integral values. Let circulation f be $\frac{1}{n+1}$-optimal wrt. some price function p. Then f is a minimum cost circulation.*

[4] Dense distance graphs have been defined differently multiple times in the literature. We use the definition of [16, 30] that captures all the known use cases (see [16] for discussion).

Proof. Suppose f is not minimum-cost. By Fact 4, f is not minimum-cost iff G_f contains a simple negative cycle C. Note that the cost of C is the same with respect to the cost functions c and c_p, as the prices cancel out. Therefore $\sum_{e\in C} c_p(e) \geq -\frac{n}{n+1} > -1$. But the cost of this cycle is integral and hence is at least 0, a contradiction. ◀

Let $C = \max_{e\in E_0}\{|c(e)|\}$. Suppose we have a procedure Refine(G, f_0, p_0, ϵ) that, given a circulation f_0 in G that is 2ϵ-optimal wrt. p_0, computes a pair (f', p') such that f' is a circulation in G, and it is ϵ-optimal wrt. p'. We use the general *scaling* framework, due to Goldberg and Tarjan [18], as given in Algorithm 1. By Lemma 6, it computes a min-cost circulation in G in $O(\log(nC))$ iterations. Therefore, if we implement Refine to run in $T(n, m)$ time, we can compute a minimum cost circulation in G in $O(T(n, m)\log(nC))$ time.

Algorithm 1 Scaling framework for min-cost circulation.

```
1: procedure MinimumCostCirculation(G)
2:     f(e) := 0 for all e ∈ G
3:     p(v) := 0 for all v ∈ V
4:     ε := C/2
5:     while ε > 1/(n+1) do                          ▷ f is 2ε-optimal wrt. p
6:         (f, p) := Refine(G, f, p, ε)
7:         ε := ε/2
8:     return f      ▷ f is circulation 1/(n+1)-optimal wrt. p, i.e., a minimum-cost circulation
```

3 Refinement via Successive Approximate Shortest Paths

In this section we introduce our implementation of Refine(G, f_0, p_0, ϵ). For simplicity, we start by setting $c(e) := c(e) - p_0(u) + p_0(v)$. After we are done, i.e., we have a circulation f' that is ϵ-optimal wrt. p', (assuming costs reduced with p_0), we will return $(f', p' + p_0)$ instead. Therefore, we now have $c(e) \geq -2\epsilon$ for all $e \in E_{f_0}$.

Let f_1 be the flow initially obtained from f_0 by sending a unit of flow through each edge $e \in E_{f_0}$ such that $c(e) < 0$. Note that f_1 is ϵ-optimal, but it need not be a circulation.

We denote by f the *current flow* which we will gradually change into a circulation. Recall that X is the set of excess vertices of G and D is the set of deficit vertices (with respect to the current flow f). Recall a well-known method of finding the min-cost circulation exactly [5, 20, 23]: repeatedly send flow through shortest $X \to D$ paths in G_f. The sets X and D would only shrink in time. However, doing this on G_f exactly would be too costly. Instead, we will gradually convert f into a circulation, by sending flow from vertices of X to vertices of D but only using approximately (in a sense) shortest paths.

Let round(y, z) denote the smallest integer multiple of z that is greater than y.

For any $e \in E$, set $c'(e) = \text{round}(c(e) + \epsilon/2, \epsilon/2)$. We define G'_f to be the "approximate" graph G_f with the costs given by c' instead of c.

For convenience, let us also define an extended version G''_f of G'_f to be G'_f with two additional vertices s (a super-excess-vertex) and t (a super-deficit-vertex) added. Let $M = \sum_{e\in E} |c'(e)| + \epsilon$. We also add to G''_f the following auxiliary edges:

1. an edge vt for all $v \in V$, we set $c'(vt) = 0$ if $v \in D$ and $c'(vt) = M$ otherwise,
2. an edge sx with $c'(sx) = 0$ for all $x \in X$.

Clearly, $\delta_{G''_f}(s, t) = \delta_{G'_f}(X, D)$ and every vertex in G''_f can reach t.

Our algorithm can be summarized very briefly, as follows. Start with $f = f_1$. While $X \neq \emptyset$, send a unit of flow along any shortest path P from X to D in G'_f (equivalently: from s to t in G''_f). Once finished, return f and $\delta_{G''_f,t}$ as the price function. The correctness of this approach follows from the following two facts that we discuss later on:

1. G'_f is negative-cycle free at all times,
2. after the algorithm finishes, f is a circulation in G that is ϵ-optimal wrt. $\delta_{G''_f,t}$.

If implemented naively, the algorithm would need $O(m)$ negative-weight shortest paths computations to finish. If we used Bellman-Ford method for computing shortest paths, the algorithm would run in $O(nm^2)$ time. To speed it up, we apply two optimizations.

First, as in the successive shortest paths algorithm for general graphs [10, 35], we observe that the distances $\delta_{G''_f,t}$ computed before sending flow through a found shortest $s \to t$ path constitute a feasible price function of G''_f *after* augmenting the flow. This allows us to replace Bellman-Ford algorithm with Dijkstra's algorithm and reduce the time to $O(m^2 + nm \log n)$. Next, instead of augmenting the flow along a single shortest $X \to D$ path, we send flow through a maximal set of edge-disjoint shortest $X \to D$ paths, as in Hopcroft-Karp algorithm for maximum bipartite matching [19]. Such a set can be easily found in $O(m)$ time when the distances to t in G''_f are known. This way, we finish after only $O(\sqrt{m})$ phases of shortest path computation and flow augmentation. The pseudocode is given in Algorithm 2.

Algorithm 2 Refinement via successive shortest paths.

Require: f_0 is a circulation in G 2ϵ-feasible wrt. p_0
Require: DistancesTo(H, t, p) computes the vector of distances (i.e., $\delta_{G,t}$) from all $v \in V(H)$ to $t \in V(H)$, where p is a feasible price function of H.
Require: SendFlow(f, E^*) returns a flow f' such that $f'(e)$ equals $f(e) + 1$ if $e \in E^*$, $f(e) - 1$ if $e^{\mathrm{R}} \in E^*$, and $f(e)$ otherwise.
Output: (f, p), where f is a circulation in G ϵ-feasible wrt. p

```
1:  procedure Refine(G, f_0, p_0, ε)
2:      c(e) := c(e) − p_0(u) + p_0(v) for all e = uv ∈ E.
3:      f := SendFlow(f_0, {e ∈ E_{f_0} : c(e) < 0})
4:      p(v) := 0 for all v ∈ V
5:      while X ≠ 0 do                          ▷ p is a feasible price function of G''_f
6:          Construct G''_f out of G'_f.
7:          p := DistancesTo(G''_f, t, p)
8:          Q_0, ..., Q_k := a maximal set of edge-disjoint s → t paths in G''_f consisting solely
                             of edges satisfying c'_p(e) = 0.
9:          f := SendFlow(f, E((Q_0 ∪ ... ∪ Q_k) ∩ G'_f))
10:     return (f, DistancesTo(G''_f, t, p) + p_0)      ▷ f is ε-feasible wrt. δ_{G''_f,t} + p_0
```

3.1 Analysis

Below we state some key properties of our refinement method. The proofs are can be found in the full version of the paper.

▶ **Lemma 7.** *Suppose G''_f has no negative cycles. Then f is ϵ-optimal wrt. $\delta_{G''_f,t}$.*

Proof. Recall that G_f and G'_f have the same sets of edges, only different costs. Let $uv = e \in G_f$. Set $p := \delta_{G''_f,t}$. By Fact 1, $c'(e) - p(u) + p(v) \geq 0$. Note that $c(e) \geq c'(e) - \epsilon$. Hence, $c(e) - p(u) + p(v) \geq c'(e) - p(u) + p(v) - \epsilon \geq -\epsilon$. ◀

▶ **Lemma 8.** *If $X \neq \emptyset$, then there exists a path from X to D in G_f.*

Before we proceed further, we need to introduce more notation. Let Δ denote the length of the shortest $X \to D$ path in G'_f (Δ changes in time along with f).

Let $q = \Psi_{f_1}$. Clearly, $q \leq m$. For $i = 1, \ldots, q$, denote by f_{i+1} the flow (with total excess $q - i$) obtained from f_i by sending a unit of flow through an arbitrarily chosen shortest $X \to D$ path P_i of G'_{f_i}.

For $i = 1, \ldots, q$, let Δ_i be the value Δ when $f = f_i$. We set $\Delta_{q+1} = \infty$.

▶ **Lemma 9.** *Let $p_i^* : V \cup \{s, t\} \to \{k \cdot \epsilon/2 : k \in \mathbb{Z}\}$ be defined as $p_i^* = \delta_{G''_{f_i}, t}$. Then:*

1. *G'_{f_i} has no cycles of non-positive cost,*
2. *for any $e \in P_i$, the reduced cost of e^{R} wrt. p_i^* is positive,*
3. *p_i^* is a feasible price function of both G''_{f_i} and $G''_{f_{i+1}}$,*
4. *$0 < \Delta_i \leq \Delta_{i+1}$.*

By Lemmas 8 and 9, our general algorithm computes a circulation f_{q+1} such that p_q^* is a feasible price function of $G'_{f_{q+1}}$. Since f_{q+1} has no negative cycles, by Lemma 7, f_{q+1} is ϵ-optimal wrt. $\delta_{G''_f, t}$. We conclude that the algorithm is correct.

The following lemma is the key to the running time analysis.

▶ **Lemma 10.** *If $X \neq \emptyset$ (equivalently, if $\Delta < \infty$), then $\Psi_f \cdot \Delta \leq 6\epsilon m$.*

3.2 Efficient Implementation

As mentioned before, we could use Lemma 9 directly: start with flow f_1 and $p_0^* \equiv 0$. Then, repeatedly compute a shortest $X \to D$ path P_i along with the values p_i^* using Dijkstra's algorithm on G''_f (with the help of price function p_{i-1}^* to make the edge costs non-negative), and send flow through P_i to obtain f_{i+1}. However, we can also proceed as in Hopcroft-Karp algorithm and augment along many shortest $X \to D$ paths of cost Δ at once. We use the following lemma.

▶ **Lemma 11.** *Let p be a feasible price function of G''_f. Suppose there is no $s \to t$ path in G''_f consisting of edges with reduced (wrt. p) cost 0. Then $\Delta = \delta_{G'_f}(X, D) > p(s) - p(t)$.*

Suppose we run the simple-minded algorithm. Assume that at some point $f = f_i$, and we have p_i^* computed. Any $s \to t$ path in G''_{f_i} with reduced (wrt. p_i^*) cost 0 corresponds to some shortest $X \to D$ path (of length Δ_i) in G'_f. Additionally, we have $p_i^*(s) = 0$ and $p_i^*(t) = \Delta_i$.

Let $Q_0, \ldots, Q_k$ be some maximal set of edge-disjoint $s \to t$ paths in G''_{f_i} with reduced cost 0. By Lemma 9, we could in principle choose $P_i = Q_0, P_{i+1} = Q_1, \ldots, P_{i+k} = Q_k$ and this would not violate the rule that we repeatedly choose shortest $X \to D$ paths.

Moreover, p_i^* is a feasible price function of $G''_{f_{i+1}}$ for any choice of $P_i = Q_j$, $j = 0, \ldots, k$. Hence, the reduced cost wrt. p_i^* of any $e^{\mathrm{R}} \in Q_j$, is non-negative. Therefore, in fact p_i^* is a feasible price function of all $G''_{f_{i+1}}, G''_{f_{i+2}}, \ldots, G''_{f_{i+k+1}}$. On the other hand, since for all $e \in P_i \cup \ldots \cup P_{i+k}$, the reduced cost (wrt. p_i^*) of e^{R} is positive, and the set $Q_0, \ldots, Q_k$ was maximal, we conclude that there is no $s \to t$ path in $G''_{f_{i+k+1}}$ consisting only of edges with reduced cost (wrt. p_i^*) 0. But $p_i^*(s) - p^*(t) = \Delta_i$, so by Lemma 11 we have $\Delta_{i+k+1} > \Delta_i$.

Since we can choose a maximal set $Q_0, \ldots, Q_k$ using a DFS-style procedure in $O(m)$ time (for details, see Section 4.3, where we take a closer look at it to implement it faster in the planar case), we can actually move from f_i to f_{i+k+1} and simultaneously increase Δ in $O(m)$ time. Since p_i^* is a feasible price function of $G''_{f_{i+k+1}}$, the new price function p_{i+k+1}^* can be computed, again, using Dijkstra's algorithm. The total running time of this algorithm is $O(m + n \log n)$ times the number of times Δ increases.

▶ **Lemma 12.** *The value Δ changes $O(\sqrt{m})$ times.*

Proof. By Lemma 9, Δ can only increase, and if it does, it increases by at least $\epsilon/2$. After it increases $2\sqrt{m}$ times, $\Delta \geq \epsilon\sqrt{m}$. But then, by Lemma 10, Ψ_f is no more than $6\sqrt{m}$. As each change of Δ is accompanied with some decrease of Ψ_f, Δ can change $O(\sqrt{m})$ times more. ◀

▶ **Theorem 13.** Refine *as implemented in Algorithm 2 runs in $O((m + n \log n)\sqrt{m})$.*

We can in fact improve the running time to $O(m\sqrt{m})$ by taking advantage of so-called Dial's implementation of Dijkstra's algorithm [9]. The details are deferred to the full version.

3.3 Bounding the Total Length of Augmenting Paths

▶ **Fact 14.** *For every $e \in E$ we have $c'(e) + c'(e^{\mathrm{R}}) > \epsilon$.*

Proof. We have $c'(e) > c(e) + \epsilon/2$. Hence, $c'(e) + c'(e^{\mathrm{R}}) > c(e) + \epsilon/2 + c(e^{\mathrm{R}}) + \epsilon/2 = \epsilon$. ◀

There is a subtle reason why we set $c'(e)$ to be $\mathrm{round}(c(e) + \epsilon/2, \epsilon/2)$ instead of $\mathrm{round}(c(e), \epsilon)$. Namely, this allows us to obtain the following bound.

▶ **Lemma 15.** *For any $i = 1, \ldots, q$ we have $c'(f_{i+1}) - c'(f_i) < \Delta_i - \frac{1}{2}|P_i| \cdot \epsilon$.*

Proof. We have

$$c'(f_{i+1}) - c'(f_i) = \frac{1}{2}\sum_{e \in E}(f_{i+1}(e) - f_i(e))c'(e) = \frac{1}{2}\sum_{e \in P_i}(c'(e) - c'(e^{\mathrm{R}})).$$

By Fact 14, $-c'(e^{\mathrm{R}}) < c'(e) - \epsilon$ for all $e \in E$. Hence

$$c'(f_{i+1}) - c'(f_i) < \sum_{e \in P_i} c'(e) - \frac{1}{2}|P_i| \cdot \epsilon = \Delta_i - \frac{1}{2}|P_i| \cdot \epsilon.$$ ◀

▶ **Lemma 16.** *Let f^* be any flow. Then $c(f_0) - c(f^*) \leq 2\epsilon m$.*

Proof. We have

$$c(f_0) - c(f^*) = \frac{1}{2}\sum_{e \in E}(f_0(e) - f^*(e))c(e).$$

If $f_0(e) > f^*(e)$, then $e^{\mathrm{R}} \in E_{f_0}$ and hence $c(e^{\mathrm{R}}) \geq -2\epsilon$, and thus $c(e) \leq 2\epsilon$. Otherwise, if $f_0(e) < f^*(e)$ then $e \in E_{f_0}$ and $c(e) \geq -2\epsilon$.

In both cases $(f_0(e) - f^*(e))c(e) \leq 2\epsilon$. Therefore, since $|E| = 2m$, $c(f_0) - c(f^*) \leq 2\epsilon m$. ◀

▶ **Lemma 17.** *Let f^* be any flow. Then $|c'(f^*) - c(f^*)| \leq \epsilon m$.*

Proof. Recall that we had $0 < c'(e) - c(e) \leq \epsilon$. Hence $|f^*(e)(c'(e) - c(e))| \leq \epsilon$ and:

$$|c'(f^*) - c(f^*)| = \frac{1}{2}\left|\sum_{e \in E} f^*(e)(c'(e) - c(e))\right| \leq \frac{1}{2}\sum_{e \in E}|f^*(e)(c'(e) - c(e))| \leq \frac{1}{2}\sum_{e \in E}\epsilon = \epsilon m.$$ ◀

The inequalities from Lemmas 15, 16 and 17 combined give us the following important property of the set of paths we augment along.

▶ **Lemma 18.** *The total number of edges on all the paths we send flow through is $O(m \log m)$.*

Proof. By Lemma 16 and the fact that $c(f_1) \leq c(f_0)$, we have:

$$c(f_1) - c(f_{q+1}) \leq c(f_0) - c(f_{q+1}) \leq 2\epsilon m.$$

On the other hand, by Lemma 17 and Lemma 15, we obtain:

$$\begin{aligned} c(f_1) - c(f_{q+1}) &\geq (c'(f_1) - \epsilon m) + (-c'(f_{q+1}) - \epsilon m) = c'(f_1) - c'(f_{q+1}) - 2\epsilon m \\ &= \sum_{i=1}^{q} (c'(f_i) - c'(f_{i+1})) - 2\epsilon m \geq \sum_{i=1}^{q} (\frac{1}{2}|P_i| \cdot \epsilon - \Delta_i) - 2\epsilon m. \end{aligned}$$

By combining the two inequalities and applying Lemma 10, we get:

$$\sum_{i=1}^{q} \frac{1}{2}|P_i| \leq 4m + \sum_{i=1}^{q} \frac{\Delta_i}{\epsilon} \leq 4m + \sum_{i=1}^{q} \frac{6m}{\Psi_{f_i}} = 4m + \sum_{i=1}^{q} \frac{6m}{q-i+1} = O(m \log m). \qquad \blacktriangleleft$$

4 Unit-Capacity Min-Cost Circulation in Planar Graphs

In this section we show that the refinement algorithm per scale from Section 3 can be simulated on a planar digraph more efficiently. Specifically, we prove the following theorem.

▶ **Theorem 19.** Refine *can be implemented on a planar graph in* $\widetilde{O}((nm)^{2/3})$ *time.*

Let $r \in [1, n]$ be a parameter. Suppose we are given an r-division with few holes $\mathcal{P}_1, \ldots, \mathcal{P}_\lambda$ of G such that for any i we have $\lambda = O(n/r)$, $|V(\mathcal{P}_i)| = O(r)$, $|\partial\mathcal{P}_i| = O(\sqrt{r})$, $\partial\mathcal{P}_i$ lies on $O(1)$ faces of $\mathcal{P}_i$, and the pieces are edge-disjoint. We set $\partial G = \bigcup_{i=1}^{\lambda} \partial\mathcal{P}_i$. Clearly, $|\partial G| = O(n/\sqrt{r})$.

In the full version we show that we can reduce our instance to the case when the above assumptions are satisfied in nearly-linear time.

Since m might be $\omega(n)$, we cannot really guarantee that $|E(\mathcal{P}_i)| = O(r)$, This will not be a problem though, since, as we will see, for all the computations involving the edges of $\mathcal{P}_i$ (e.g., computing shortest paths in $\mathcal{P}_i$, or sending a unit of flow through a path of $\mathcal{P}_i$) of all edges $uv = e \in E(\mathcal{P}_i)$ we will only care about an edge $e \in E(\mathcal{P}_i) \cap G_f$ with minimal cost $c'(e)$. Therefore, since $\mathcal{P}_i$ is planar, at any time only $O(r)$ edges of $\mathcal{P}_i$ will be needed.

Recall that the per-scale algorithm for general graphs (Algorithm 2) performed $O(\sqrt{m})$ phases, each consisting of two steps: a shortest path computation (to compute the price function p^* from Lemma 9), followed by the computation of a maximal set of edge-disjoint augmenting paths of reduced (wrt. p^*) cost 0. We will show how to implement both steps in $\widetilde{O}(n/\sqrt{r})$ amortized time, at the additional total data structure maintenance cost (over all phases) of $\widetilde{O}(mr)$. Since there are $O(\sqrt{m})$ steps, this will yield $\widetilde{O}(nm)^{2/3})$ time by appropriately setting r.

We can maintain the flow f explicitly, since it undergoes only $O(m \log n)$ edge updates (by Lemma 18). However, we will not compute the entire price function p^* at all times explicitly, as this is too costly. Instead, we will only compute p^* limited to the subset $\partial G \cup \{s, t\}$.

For each $\mathcal{P}_i$, define $\mathcal{P}'_{f,i} = G'_f \cap \mathcal{P}_i$. We also define $\mathcal{P}''_{f,i}$ to be $\mathcal{P}'_{f,i}$ with vertices $\{s, t\}$ added, and those edges sv, vt of G''_f that satisfy $v \in V(\mathcal{P}_i) \setminus \partial\mathcal{P}_i$. This way, $\mathcal{P}''_{f,i} \subseteq G''_f$ and $E(\mathcal{P}''_{f,i}) \cap E(\mathcal{P}''_{f,j}) = \emptyset$ for $i \neq j$. The costs of edges $e \in E(\mathcal{P}''_{f,i})$ are the same as in G''_f, i.e., $c'(e)$. Besides, for each i we will store a "local" price function p_i that is feasible only for $\mathcal{P}''_{f,i}$,

After the algorithm finishes, we will know how the circulation looks like precisely. However, the general scaling algorithm requires us to also output price function p such that f is an ϵ-optimal circulation wrt. p. f is ϵ-optimal wrt. p^* in the end, but we will only have it computed for the vertices $\partial G \cup \{s, t\}$. Therefore, we extend it to all remaining vertices of G.

▶ **Lemma 20.** *Suppose we are given the values of p^* on $\partial\mathcal{P}_i$ and a price function p_i feasible for $\mathcal{P}''_{f,i}$. Then we can compute the values $p^*(u)$ for all $v \in V(\mathcal{P}''_{f,i})$ in $O(r \log r)$ time.*

Hence, in order to extend p^* to all vertices of G once the final circulation is found, we apply Lemma 20 to all pieces. This takes $O\left(\frac{n}{r} \cdot r \log r\right) = O(n \log n)$ time.

4.1 Dijkstra Step

Let us start with an implementation of the Dijkstra step computing the new price function p^*. First, for each piece $\mathcal{P}_i$ we define the compressed version $H''_{f,i}$ of $\mathcal{P}''_{f,i}$ as follows. Let $V(H''_{f,i}) = \partial\mathcal{P}_i \cup \{s,t\}$. The set of edges of $H''_{f,i}$ is formed by:

- a distance clique $\mathrm{DC}(\mathcal{P}''_{f,i})$ between vertices $\partial\mathcal{P}_i$ in $\mathcal{P}''_{f,i}$,
- for each $v \in \partial\mathcal{P}_i$, an edge sv of cost $\delta_{\mathcal{P}''_{f,i}}(s,v)$ if this distance is finite,
- for each $v \in \partial\mathcal{P}_i$, an edge vt of cost $\delta_{\mathcal{P}''_{f,i}}(v,t)$ if this distance is finite,
- an edge st of cost $\delta_{\mathcal{P}''_{f,i}}(s,t)$ if this distance is finite.

Recall that we store a price function p_i of $\mathcal{P}''_{f,i}$. Therefore, by Theorem 2, $\mathrm{DC}(\mathcal{P}''_{f,i})$ can be computed in $O(r \log r)$ time. All needed distances $\delta_{\mathcal{P}''_{f,i}}(s,v)$ and $\delta_{\mathcal{P}''_{f,i}}(v,t)$ can be computed in $O(r \log r)$ time using Dijkstra's algorithm (again, with the help of price function p_i).

Now define H''_f to be $\bigcup_{i=1}^{\lambda} H''_{f,i}$ with edges sv and vt of G''_f that satisfy $v \in \partial G$ added.

▶ **Fact 21.** *For any $u, v \in V(H''_f)$, $\delta_{H''_f}(u,v) = \delta_{G''_f}(u,v)$.*

Observe that H''_f is a dense distance graph in terms of the definition of Section 2: it consists of $O(n/r)$ distance cliques $\mathrm{DC}(\mathcal{P}''_{f,i})$ with $O(\sqrt{r})$ vertices each, and $O(n/\sqrt{r})$ additional edges which also can be interpreted as 2-vertex distance cliques.

Hence, given a feasible price function on H''_f, we can compute distances to t in H''_f on it using Theorem 3 in $O\left(n/\sqrt{r}\frac{\log^2 n}{\log^2 \log n}\right)$ time. Since $V(H''_f) = \partial G \cup \{s,t\}$, the price function p^* we have is indeed sufficient. The computed distances to t form the new price function p^* on $\partial G \cup \{s,t\}$ as in the algorithm for general graphs (see Algorithm 2).

4.2 Sending Flow Through a Path

In the general case updating the flow after an augmenting path has been found was trivial. However, as we operate on a compressed graph, the update procedure has to be more involved.

Generally speaking, we will repeatedly find some shortest $s \to t$ path $Q = e_1 \dots e_k$ in H''_f, translate it to a shortest $s \to t$ path $\mathcal{P}$ in G''_f and send flow through it. It is easy to see by the definition of H''_f that Q can be translated to a shortest $s \to t$ path in G''_f and vice versa. Each edge e_j can be translated to either some subpath inside a single graph $\mathcal{P}''_{f,i}$, or an edge of G''_f of the form sv or vt, where $v \in \partial G$. This can be done in $O(r \log n)$ time by running Dijkstra's algorithm on $\mathcal{P}''_{f,i}$ with price function p_i. We will guarantee that path P obtained by concatenating the translations of individual edges e_j contains no repeated edges of G''_f.

We now show how to update each $H''_{f,i}$ after sending flow through the found path P. Note that we only need to update $H''_{f,i}$ if $E(P) \cap E(\mathcal{P}''_{f,i}) \neq \emptyset$. In such case we call $\mathcal{P}_i$ an *affected piece*. Observe that some piece can be affected at most $O(m \log m)$ times since the total number of edges on all shortest augmenting paths P in the entire algorithm, regardless of their choice, is $O(m \log m)$ (see Lemma 18).

To rebuild $H''_{f,i}$ to take into account the flow augmentation we will need a feasible price function on $\mathcal{P}''_{f,i}$ after the augmentation. However, we cannot be sure that what we have, i.e., p_i, will remain a good price function of $\mathcal{P}''_{f,i}$ after the augmentation. By Lemma 9, luckily, we know that p^* is a feasible price function after the augmentation for the whole

graph G''_f. In particular, p^* (before the augmentation) limited to $V(\mathcal{P}''_{f,i})$ is a feasible price function of $\mathcal{P}''_{f,i}$ after the augmentation. Hence, we can compute new p_i equal to p^* using Lemma 20 in $O(r \log r)$ time. Given a feasible price function p_i on $\mathcal{P}''_{f,i}$ after f is augmented, we can recompute $H''_{f,i}$ in $O(r \log r)$ time as discussed in Section 4.1. We conclude that the total time needed to update the graph H''_f subject to flow augmentations is $O(mr \log r \log m) = O(mr \log n \log m)$.

4.3 A Path Removal Algorithm

In this section we consider an abstract "path removal" problem, that generalizes the problem of finding a maximal set of edge-disjoint $s \to t$ paths. We will use it to reduce the problem of finding such a set of paths on a subgraph of G''_f consisting of edges with reduced cost 0 wrt. p^* to the problem of finding such a set of paths on the zero-reduced cost subgraph of H''_f.

Suppose we have some directed acyclic graph H with a fixed source s and sink t, that additionally undergoes some limited adversarial changes. We are asked to efficiently support a number of *rounds*, until t ceases to be reachable from s. Each round goes as follows.

1. We first find either any $s \to t$ path P, or detect that no $s \to t$ path exists.
2. Let $E^+ \subseteq V \times V$, and $P \subseteq E^- \subseteq E(H)$ be some adversarial sets of edges. Let $H' = (V, E')$, where $E' = E(H) \setminus E^- \cup E^+$. Assume that for any $v \in V(H)$, if v cannot reach t in H, then v cannot reach t in H' either. Then the adversarial change is to remove E^- from E and add E^+ to E, i.e., set $E(H) = E'$.

Let $\bar{n} = |V(H)|$ and let $\bar{m}$ be the number of edges ever seen by the algorithm, i.e., the sum of $|E(H)|$ and all $|E^+|$. We will show an algorithm that finds all the paths P in $O(\bar{n}+\bar{m})$ total time. Let us also denote by $\bar{\ell}$ the sum of lengths of all returned paths P. Clearly, $\bar{\ell} \leq \bar{m}$.

A procedure handling the phase 1 of each round, i.e., finding a $s \to t$ path or detecting that there is none, is given in Algorithm 3. The second phase of each round simply modifies the representation of the graph H accordingly. Throughout all rounds, we store a set W of vertices w of H for which we have detected that there is no more $w \to t$ path in H. Initially, $W = \emptyset$. Each edge $e \in E(H)$ can be *scanned* or *unscanned*. Once e is scanned, it remains scanned forever. The adversarial edges E^+ that are inserted to $E(H)$ are initially unscanned. The following lemmas establishing the correctness and efficiency of the crucial parts of Algorithm 3 are all proved in the full version.

Algorithm 3 Path-finding procedure. Returns a $s \to t$ path in H or detects there is none.

```
1:  procedure FindPath(H)
2:      Q := an empty path with a single endpoint s          ▷ Q is an s → s path
3:      while s ∉ W and the other endpoint y of Q is not equal to t do   ▷ Q is an s → y path
4:          if there exists an unscanned edge yv = e ∈ E(H) such that v ∉ W then
5:              mark e scanned
6:              Q := Qe
7:          else
8:              W := W ∪ {y}
9:              remove the last edge of Q unless Q is empty
10:     if Q = ∅ then
11:         report t not reachable from s and stop
12:     else
13:         return Q and Q := 0.
```

▶ **Lemma 22.** *Algorithm 3 correctly finds an $s \to t$ path in H or detects there is none.*

▶ **Lemma 23.** *The total number of times line 9 is executed, through all rounds, is $O(\bar{n})$.*

▶ **Lemma 24.** *Line 6 of Algorithm 3 is executed $O(\bar{n} + \bar{\ell})$ times through all rounds.*

▶ **Lemma 25.** *The total time used by Algorithm 3, through all rounds, is $O(\bar{n} + \bar{m})$.*

4.4 Finding a Maximal Set of Shortest Augmenting Paths

Recall that for a general graph, after computing the price function p^* we found a maximal set of edge-disjoint $s \to t$ paths in the graph Z''_f, defined as a subgraph of G''_f consisting of edges with reduced cost 0 (wrt. p^*). To accomplish that, we could in fact use the path removal algorithm from Section 4.3 run on Z''_f: until there was an $s \to t$ path in Z''_f, we would find such a path P, remove edges of P (i.e., set $E^- = P$ and $E^+ = \emptyset$), and repeat. Since in this case we never add edges, the assumption that t cannot become reachable from any v due to updating Z''_f is met.

Let Y''_f be the subgraph of the graph H''_f from Section 4.1 consisting of edges with reduced (wrt. p^*) cost 0. Since all edges of H''_f correspond to shortest paths in G''_f, all edges of Y''_f correspond to paths in G''_f with reduced cost 0. Because Z''_f is acyclic by Lemma 9, Y''_f is acyclic as well. Moreover, for any two edges $e_1, e_2 \in E(Y''_f)$, if there is a path going through both e_1 and e_2 in Y''_f, then the paths represented by e_1 and e_2 are edge-disjoint in Z''_f (as otherwise Z''_f would have a cycle). Therefore, any path Q in Y''_f translates to a *simple* path in $Z''_f \subseteq G''_f$.

We will now explain why running Algorithm 3 on Y''_f can be used to find a maximal set of edge-disjoint $s \to t$ paths. Indeed, by Fact 21, Y''_f contains an $s \to t$ path iff Z''_f does. Since Y''_f is just a compressed version of Z''_f, and Z''_f undergoes edge deletions only (since we only remove the found paths), the updates to Y''_f cannot make some t reachable from some new vertex $v \in V(Y''_f)$. Technically speaking, we should think of Y''_f as undergoing both edge insertions and deletions: whenever some path $Q \subseteq Y''_f$ is found, we include Q in E^- and send the flow through a path corresponding to Q in G''_f, as described in Section 4.2. But then for all affected pieces $\mathcal{P}_i$, $H''_{f,i}$ is recomputed and thus some of the edges of Q might be reinserted to Y''_f again. These edges should be seen as forming the set E^+, whereas the old edges of the recomputed graphs $H''_{f,i}$ belong to E^-. In terms of the notation of Section 4.3, when running Algorithm 3 on Y''_f, we have $\bar{n} = O(n/\sqrt{r})$. The sum of values $\bar{\ell}$ from Section 4.3 over all phases of the algorithm is, by Lemma 18, $O(m \log m)$. Similarly, again by Lemma 18, the sum of the values $\bar{m}$ from Section 4.3 over all phases, is $O(m^{3/2} + mr^2 \log m)$ (since each time E^+ might be as large as r^2 times the number of affected pieces).

Recall that there are $O(\sqrt{m})$ phases, and the total time needed to maintain the graph H''_f subject to flow augmentations is $O(mr \log r \log m)$ (see Section 4.2). For each phase, running a Dijkstra step to compute p^* using FR-Dijkstra, followed by running Algorithm 3 directly until there are no $s \to t$ paths in Y''_f would lead to $O\left(\sqrt{m}\left(\frac{n}{\sqrt{r}}\frac{\log^2 n}{\log^2 \log n}\right) + m^{3/2} + mr^2 \log m\right)$ total time, i.e., would not yield any improvement over the general algorithm. However, we can do better by implementing Algorithm 3 on Y''_f more efficiently.

▶ **Lemma 26** ([21, 24])**.** *Let Z be the subgraph of $\mathcal{P}'_{f,i}$ consisting of edges with reduced cost 0 with respect to some feasible price function p. There exists $O(\sqrt{r})$ pairs of subsets $(A_{i,1}, B_{i,1}), (A_{i,2}, B_{i,2}), \ldots$ of $\partial\mathcal{P}_i$ such that for each $v \in \partial\mathcal{P}_i$:*

- *The number of sets $A_{i,j}$ ($B_{i,j}$) such that $v \in A_{i,j}$ ($v \in B_{i,j}$, resp.) is $O(\log r)$.*
- *Each $B_{i,j}$ is totally ordered according to some order $\prec_{i,j}$.*

- *For any j such that $v \in A_{i,j}$, there exist $l_{i,v,j}, r_{i,v,j} \in B_{i,j}$ such that the subset $R_{i,v}$ of $\partial\mathcal{P}_i$ reachable from v in Z can be expressed as $\bigcup_{j:v\in A_{i,j}}\{w \in B_{i,j} : l_{i,v,j} \preceq_{i,j} w \preceq_{i,j} r_{i,v,j}\}$.*

The sets $A_{i,j}$, $B_{i,j}$ and the vertices $l_{i,v,j}, r_{i,v,j}$ for all v, j can be computed in $O(\sqrt{r}\log r)$ time based on the distance clique between $\partial\mathcal{P}_i$ in $\mathcal{P}'_{f,i}$ and the values of p^* on $\partial\mathcal{P}_i$.

Recall that in Section 4.3, to bound the total running time, it was enough to bound the total time spent on executing lines 4, 6 and 9. We will show that using Lemma 26, in terms of the notation from Section 4.3, we can make the total time spent on executing line 4 only $\widetilde{O}(\bar{n}+\bar{\ell})$ instead of $O(\bar{m})$, at the cost of increasing the total time of executing line 9 to $\widetilde{O}(\bar{n})$.

Specifically, at the beginning of each phase we compute the data from Lemma 26 for all pieces $\mathcal{P}_i$. Since for all i we have the distance cliques $\mathrm{DC}(\mathcal{P}''_{f,i})$ computed, this takes $O\big(\frac{n}{r}\cdot\sqrt{r}\log r\big) = O(n/\sqrt{r}\log n)$ time. We will also recompute the information of Lemma 26 for an affected piece $\mathcal{P}_i$ after $H''_{f,i}$ is recomputed. As the total number of times some piece is affected is $O(m\log m)$, this takes $O(m\sqrt{r}\log r\log m)$ time through all phases.

Whenever the data of Lemma 26 is computed for some piece $\mathcal{P}_i$, for each pair $(A_{i,j}, B_{i,j})$ we store $B_{i,j}\cap W$ in a dynamic predecessor/successor data structure $D_{i,j}$, sorted by $\prec_{i,j}$. For each $v \in \partial\mathcal{P}_i$ and j such that $v \in A_{i,j}$ we store a vertex $next_{i,v,j}$ initially equal to $l_{i,v,j}$. It is easy to see that these auxiliary data structures can be constructed in time linear in their size, i.e., $O(\sqrt{r}\log r)$ time. Hence, the total cost of computing them is $O(\sqrt{m}n/\sqrt{r}\log n + m\sqrt{r}\log r\log m) = O\Big(\sqrt{m}\Big(\frac{n}{\sqrt{r}}\frac{\log^2 n}{\log^2\log n}\Big) + mr\log n\log m\Big)$.

Now, to implement line 9, when y is inserted into W we go through all pieces $\mathcal{P}_i$ such that $y \in \partial\mathcal{P}_i$ and all $B_{i,j}$ such that $y \in B_{i,j}$. For each such (i,j), we remove y from $D_{i,j}$ in $O(\log\log n)$ time. Recall that the sum of numbers of such pairs (i,j) over all $v \in \partial G$ is $O(\sum_{i=1}^{\lambda}|\partial\mathcal{P}_i|\log r) = O(n/\sqrt{r}\log n)$. Hence, by Lemma 23 the total time spent on executing line 9 in a single phase is $O(n/\sqrt{r}\log n\log\log n)$.

Finally, we implement line 4 as follows. The unscanned edges of Y''_f that are not between boundary vertices are handled in a simple-minded way as in Lemma 25. There are only $O(n/\sqrt{r})$ of those, so we can neglect them. In order to be able to efficiently find some unscanned edge yv such that $y, v \in \partial G$ and $v \notin W$, we keep for any $v \in \partial G$ a set U_v of pieces $\mathcal{P}_i$ such that $v \in \partial\mathcal{P}_i$ and there may still be some unscanned edges from v to $w \in \partial\mathcal{P}_i$ in $H''_{f,i}$. Similarly, for each $\mathcal{P}_i \in U_v$ we maintain a set $U_{v,i}$ of data structures $D_{i,j}$ such that $next_{i,v,j} \neq \mathbf{nil}$. Whenever the data of Lemma 26 is computed for $\mathcal{P}_i$, $\mathcal{P}_i$ is inserted back to U_v for all $v \in \partial\mathcal{P}_i$, and the sets $U_{v,i}$ are recomputed with no additional asymptotic overhead. To find an unscanned edge yv, for each $\mathcal{P}_i \in U_y$ we proceed as follows. We attempt to find an unscanned edge yv in $\mathcal{P}_i$. If we succeed or U_y is empty, we stop. Otherwise we remove $\mathcal{P}_i$ from U_y and repeat, i.e., try another $\mathcal{P}_j \in U_y$, unless U_y is empty. To find an unscanned edge yv from a piece $\mathcal{P}_i$, we similarly try to find an unscanned edge yv in subsequent data structures $D_{i,j} \in U_{v,i}$, and remove the data structures for which we fail from $U_{v,i}$. For a single data structure $D_{i,j}$, we maintain an invariant that an edge yw, $w \in D_{i,j}$ has been scanned iff $w \prec_{i,j} next_{i,v,j}$. Hence, to find the next unscanned edge, we first find $x \in D_{i,j}$ such that $next_{i,v,j} \preceq_{i,j} x$ and x is smallest possible. This can be done in $O(\log\log n)$ time since $D_{i,j}$ is a dynamic successor data structure. If x does not exist or $r_{i,v,j} \prec x$, then, by Lemma 26, there are no more unscanned edges yw such that $w \in D_{i,j}$, and thus we remove $D_{i,j}$ from $U_{v,i}$. Otherwise, we return an edge yx and set $next_{i,v,j}$ to be the successor of x in $D_{i,j}$ (or possibly $next_{i,v,j} := \mathbf{nil}$ if none exists), again in $O(\log\log n)$ time.

Observe that all "failed" attempts to find an edge yv, where $v \in \partial G$ can be charged to an insertion of some $\mathcal{P}_i$ to U_y or to an insertion of some $D_{i,j}$ to $U_{y,i}$. The total number of such insertions is again $O\Big(\sqrt{m}\frac{n}{\sqrt{r}}\log n + m\sqrt{r}\log r\log m\Big)$. A successful at-

tempt, on the other hand, costs $O(\log \log n)$ worst-case time. Since line 4 is executed $O(\sqrt{m}n/\sqrt{r} + m \log n)$ times through all phases, the total time spent on executing line 4 is again $O\left(\sqrt{m}\left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log^2 \log n}\right) + mr \log n \log m\right)$. By setting $r = \frac{n^{2/3}}{m^{1/3}} \cdot \left(\frac{\log n}{\log m \cdot \log^2 \log n}\right)^{2/3}$ we obtain the main result of this paper.

▶ **Theorem 27.** *The min-cost circulation in a planar multigraph can be found in* $O\left((nm)^{2/3} \cdot \frac{\log^{5/3} n \log^{1/3} m}{\log^{4/3} \log n} \cdot \log (nC)\right)$ *time.*

References

1. Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A Faster Algorithm for Minimum-Cost Bipartite Perfect Matching in Planar Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 457–476, 2018. doi:10.1137/1.9781611975031.31.
2. Dimitri P. Bertsekas and Paul Tseng. Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems. *Operations Research*, 36(1):93–114, 1988. doi:10.1287/opre.36.1.93.
3. 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.
4. 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.
5. Robert G Busacker and Paul J Gowen. A procedure for determining a family of minimum-cost network flow patterns. Technical report, RESEARCH ANALYSIS CORP MCLEAN VA, 1960.
6. 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.
7. Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}$ ($m^{10/7}$ log W) Time (Extended Abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771, 2017. doi:10.1137/1.9781611974782.48.
8. Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 451–460, 2008. doi:10.1145/1374376.1374441.
9. Robert B. Dial. Algorithm 360: shortest-path forest with topological ordering [H]. *Commun. ACM*, 12(11):632–633, 1969. doi:10.1145/363269.363610.
10. Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.
11. Jeff Erickson. Maximum Flows and Parametric Shortest Paths in Planar Graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804, 2010. doi:10.1137/1.9781611973075.65.
12. Shimon Even and Robert Endre Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. doi:10.1137/0204043.
13. 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.
14. L. R. Ford and D. R. Fulkerson. Constructing Maximal Dynamic Flows from Static Flows. *Operations Research*, 6(3):419–433, 1958.

15 Harold N. Gabow and Robert Endre Tarjan. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989. doi:10.1137/0218069.

16 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, July 9-13, 2018, Prague, Czech Republic*, pages 61:1–61:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.61.

17 Andrew V. Goldberg, Sagi Hed, Haim Kaplan, and Robert E. Tarjan. Minimum-Cost Flows in Unit-Capacity Networks. *Theory Comput. Syst.*, 61(4):987–1010, 2017. doi:10.1007/s00224-017-9776-7.

18 Andrew V. Goldberg and Robert E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. Oper. Res.*, 15(3):430–466, 1990. doi:10.1287/moor.15.3.430.

19 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

20 Masao Iri. A new method of solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 1960.

21 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, 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, Montreal, QC, Canada, June 19-23, 2017*, pages 1108–1121, 2017. doi:10.1145/3055399.3055480.

22 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, San Jose, CA, USA, 6-8 June 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

23 William S. Jewell. Optimal Flow through Networks with Gains. *Operations Research*, 10(4):476–499, 1962. URL: http://www.jstor.org/stable/168050.

24 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, New Orleans, LA, USA, January 7-10, 2018*, pages 73–92, 2018. doi:10.1137/1.9781611975031.5.

25 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

26 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, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

27 Nathaniel Lahn and Sharath Raghvendra. A Faster Algorithm for Minimum-Cost Bipartite Matching in Minor-Free Graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 569–588, 2019. doi:10.1137/1.9781611975482.36.

28 Yin Tat Lee and Aaron Sidford. Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}$(vrank) Iterations and Faster Algorithms for Maximum Flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.

29 Gary L. Miller and Joseph Naor. Flow in Planar Graphs with Multiple Sources and Sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. doi:10.1137/S0097539789162997.

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

31 James B. Orlin. A Faster Strongly Polynominal Minimum Cost Flow Algorithm. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 377–387, 1988. doi:10.1145/62212.62249.

32 Thomas L Saaty and Robert G Busacker. *Finite graphs and networks: An introduction with applications*. McGraw-Hill Book Company, 1965.

33 Piotr Sankowski. NC algorithms for weighted planar perfect matching and related problems. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 97:1–97:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.97.

34 Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985. doi:10.1007/BF02579369.

35 Nobuaki Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.