SIAM J. Comput.
Vol. 46, No. 4, pp. 1280–1303

MULTIPLE-SOURCE MULTIPLE-SINK MAXIMUM FLOW IN DIRECTED PLANAR GRAPHS IN NEAR-LINEAR TIME*

GLENCORA BORRADAILE†, PHILIP N. KLEIN‡, SHAY MOZES§, YAHAV NUSSBAUM¶, AND CHRISTIAN WULFF-NILSEN‖

Abstract. We give an $O(n \log^3 n)$ algorithm that, given an n-node directed planar graph with arc capacities, a set of source nodes, and a set of sink nodes finds a maximum flow from the sources to the sinks. Previously, the fastest algorithms known for this problem were those for general graphs.

Key words. planar graphs, maximum flow

AMS subject classifications. 05C10, 05C21

DOI. 10.1137/15M1042929

1. Introduction.

The maximum flow problem with multiple sources and sinks in a directed graph with arc-capacities is, informally, to find a way to route a single commodity from a given set of sources to a given set of sinks such that the total amount of the commodity that is delivered to the sinks is maximum subject to each arc carrying no more than its capacity. In this paper we study this problem in planar graphs.

The study of maximum st-flow in planar graphs, when there is one source s and one sink t, has a long history. In 1956, Ford and Fulkerson introduced the max st-flow problem, gave a generic augmenting-path algorithm, and also gave a particular augmenting-path algorithm for the case of a planar graph where s and t are on the same face. Researchers have since published many algorithmic results proving running-time bounds on max st-flow for (a) planar graphs where s and t are on the same face, (b) undirected planar graphs where s and t are arbitrary, and (c) directed planar graphs where s and t are arbitrary. The best bounds known are (a) $O(n)$ [20], (b) $O(n \log \log n)$ [24], and (c) $O(n \log n)$ [3], where n is the number of nodes in the graph.

Maximum flow in planar graphs with multiple sources and sinks was studied by Miller and Naor [38]. They gave a divide-and-conquer algorithm for the case where all the sinks and the sources are on the boundary of a single face. Plugging in the linear-time shortest-path algorithm of Henzinger et al. [20] yields a running time of

*Received by the editors October 6, 2015; accepted for publication (in revised form) May 10, 2017; published electronically August 1, 2017. A preliminary version of this work appeared in *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science* (*FOCS*) 2011, pp. 170–179.
http://www.siam.org/journals/sicomp/46-4/M104292.html

Funding: This work is partially supported by the US National Science Foundation, under grant CCF-0963921, grant CCF-0964037, and grant CCF-1409520, by the Israeli Science Foundation, under grant 822/10 and grant 974/13, and by the Israeli Centers of Research Excellence (I-CORE) program (Center 4/11).

†School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 (glencora@eecs.orst.edu).

‡Department of Computer Science, Brown University, Providence, RI 02912 (klein@brown.edu).

§Efi Arazi School of Computer Science, the Interdisciplinary Center, Herzliya 46150, Israel (smozes@idc.ac.il).

¶Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (yahav.nussbaum@cs.tau.ac.il).

‖Department of Computer Science, University of Copenhagen, 1017 Copenhagen, Denmark (koolooz@di.ku.dk).

$O(n \log n)$. Miller and Naor also gave an algorithm for the case where the sources and the sinks reside on the boundaries of k different faces. Plugging in the the $O(n \log n)$-time single-source single-sink maximum flow algorithm of Borradaile and Klein [3] yields a running time of $O(k^2 n \log^2 n)$. Miller and Naor show that, when it is known how much of the commodity is produced/consumed at each source and each sink, finding a consistent routing of flow that respects arc capacities can be reduced to *negative-length shortest paths* [38], which can be solved in planar graphs in $O(n \log^2 n / \log \log n)$ time [39].

However, the problem of maximum flow with multiple sources and sinks in planar graphs without any additional restrictions remained open. In general (i.e., nonplanar) graphs, multiple sources and sinks can be reduced to the single-source single-sink case by introducing an artificial source and sink and connecting them to all the sources and sinks, respectively—but this reduction does not preserve planarity. For more than twenty years since the problem was explicitly stated and considered [38], the fastest known algorithm for computing multiple-source multiple-sink max-flow in a planar graph has been to use this reduction in conjunction with a general maximum-flow algorithm. Using Orlin's algorithm for sparse graphs [41] leads to a running time of $O(n^2 / \log n)$. For integer capacities less than U, one could instead use the algorithm of Goldberg and Rao [16], which leads to a running time of $O(n^{1.5} \log n \log U)$. No planarity-exploiting algorithm was known for the problem.

Our result. The main result of this paper is an algorithm for the problem that is optimal up to a small poly-logarithmic factor.

THEOREM 1.1. *There exists an algorithm that solves the maximum flow problem with multiple sources and sinks in an n-node directed planar graph with arc capacities in $O(n \log^3 n)$ time.*

In section 5 we use our main result to show that there is an algorithm that solves the maximum flow problem in an n-node directed planar graph with multiple sources and sinks and with k nodes with finite capacity in $O(k^3 n \log^3 n)$ time. We also show that there is an algorithm that solves the maximum flow problem in a directed k-apex graph in $O(k^3 n \log^3 n)$ time.

Application to computer vision problems. Via the max flow, min cut theorem [12], solving a maximum flow problem also solves the corresponding minimum cut problem. Multiple-source multiple-sink min-cut arises in addressing a family of problems associated with the terms *metric labeling* [34], *Markov random fields* [15], and *Potts model* (see also [5, 21]). In low-level vision problems such as *image restoration*, *segmentation*, *stereo*, and *motion*, the goal is to assign labels from a set to pixels so as to minimize a penalty function. The penalty function is a sum of two parts. One part, the *data component*, has a term for each pixel; the cost depends on the discrepancy between the observed data for the pixel and the label chosen for it. The other part, the *smoothing component*, penalizes neighboring pixels that are assigned different labels.

For the *binary* case (when the set of available labels has size two), finding the optimal solution is reducible to multiple-source multiple-sink min-cut [18]. For the case of more than two labels, there is a powerful and effective heuristic [5] using very-large-neighborhood [1] local search; the inner loop consists of solving the two-label case. The running time for solving the two-label case is therefore quite important. For this reason, researchers in computer vision have proposed new algorithms for max flow and performed experimental studies comparing the run-times of different max-flow algorithms on the instances arising in this context [4, 35].

For the (common) case where the underlying graph of pixels is the two-dimensional grid graph, our result yields a theoretical speed-up for this important computer-vision subroutine. The single-source, single-sink max-flow algorithm of Borradaile and Klein [3] was implemented by computer-vision researchers [42] and found to be useful in computer vision and to be faster than competing algorithms.

Hochbaum [21] describes a special case of the penalty function in which the data component is convex and the smoothing component is linear; in this case, she shows that an optimal solution can be found in time $O(T(m,n) + n \log U)$, where U is the maximum label and $T(m,n)$ is the time for finding a minimum cut in graph with m edges and n vertices. She specifically mentions image segmentation, for which the graph is planar.

Other applications. It is well-known how to reduce *maximum matching in a bipartite planar graph* to multiple-source, multiple-sink maximum flow. Our result is the first planarity-exploiting (and first near-linear time) algorithm for this problem. Multiple-source, multiple-sink maximum flow can also be used for finding *orthogonal drawings of planar graphs with a minimum number of bends* [8] and *uniformly monotone subdivisions of polygons* [45].

Techniques. To obtain our result, we employ a wide range of sophisticated algorithmic techniques for planar graphs, some of which we adapted to our needs while others are used unchanged. Our algorithm uses pseudoflows [17, 22] and a divide-and-conquer scheme influenced by that of Johnson and Venkatesan [26] and that of Miller and Naor [38]. We adapt a method for using shortest paths to solve max st-flow when s and t are adjacent [19] and a data structure for implementing Dijkstra's algorithm in a dense distance graph derived from a planar graph [11]. Among the other techniques we employ are using cycle separators [37] recursively while keeping the boundary nodes on a constant number of faces [33, 44, 11], an algorithm for single-source single-sink maximum flow [3, 10], an algorithm for computing multiple-source shortest paths [31, 6], a method for cancelling cycles of flow in a planar graph [28], an algorithm for finding shortest paths in planar directed graphs with negative lengths [32, 39], and a data structure for range queries in a Monge matrix [27, 13, 14].

One of the important new tools we develop is an efficient procedure for adjusting a pseudoflow so as to ensure that there is no residual path from a vertex with positive inflow on a path P to a vertex with positive outflow on P. This procedure has recently found uses in efficient algorithms for other related problems. It is used in a nearly linear time algorithm for computing single-source all sinks maximum flows in planar graphs [36], and in a nearly linear time algorithm for constructing a data structure capable of reporting, in constant time, the weight of any minimum st-cut in a surface embedded graph [2].

1.1. Preliminaries.

Planar graphs are those that can be drawn on the surface of a sphere such that vertices map to points and edges map to line segments connecting images of vertices and such that these line segments only intersect at vertices. The connected regions of the sphere formed by the complement of the vertices and edges are the *faces* of the graph. This is a geometric embedding. For the purposes of this work, we need only a combinatorial embedding for a planar embedded graph which, for every vertex, defines the cyclic order in which the edges adjacent to that vertex are embedded.

For every planar embedded graph, there is a dual graph G^* whose vertices are the faces of G and whose edges connect vertices whose corresponding faces in G share a boundary.

Let $G = (V, E)$ be a planar embedded graph with node-set V and arc-set E. We use the term *arc* to emphasize that edges are directed. The term *edge* is used when the direction of an arc is not important. For each arc a in the arc-set E, we define two oppositely directed *darts*, one in the same orientation as a (which we sometimes identify with a) and one in the opposite orientation [3]. We define $\text{rev}(\cdot)$ to be the function that maps each dart to the corresponding dart in the opposite direction. Since edges, arcs, and darts of G are in a one to one correspondence with edges, arcs, and darts of the dual G^*, respectively, it is notationally convenient to equate the edges, arcs, and darts of G with the edges, arcs, and darts of G^*. For an arc $e = uv$, we say that u is the *tail* of e and v is the *head* of v and denote $u = \text{tail}(e), v = \text{head}(e)$. The same definition applies to darts. For a path P we define $\text{start}(P)$ and $\text{end}(P)$ to be the first and last vertex in P, respectively.

Let $S \subseteq V$ be a set of nodes called sources, and let $T \subseteq V - S$ be a set of nodes called sinks.

A *flow assignment* $\boldsymbol{f}(\cdot)$ is a real-valued function on darts satisfying *antisymmetry*:

$$\boldsymbol{f}(\text{rev}(d)) = -\boldsymbol{f}(d).$$

A *capacity assignment* $\boldsymbol{c}(\cdot)$ is a real-valued function on darts.[1] A flow assignment $\boldsymbol{f}(\cdot)$ *respects the capacity* of dart d if $\boldsymbol{f}(d) \leq \boldsymbol{c}(d)$. We call $\boldsymbol{f}(\cdot)$ a *pseudoflow* if it respects the capacities of all darts.

For a given flow assignment $\boldsymbol{f}(\cdot)$, the *net inflow* (or just *inflow*) of node v is $\text{inflow}_{\boldsymbol{f}}(v) = \sum_{\text{dart } d:\text{head}(d)=v} \boldsymbol{f}(d)$. An equivalent definition, in terms of arcs, is $\text{inflow}_{\boldsymbol{f}}(v) = \sum_{a \in E:\text{head}(a)=v} \boldsymbol{f}(a) - \sum_{a \in E:\text{tail}(a)=v} \boldsymbol{f}(a)$. The *outflow* of v is defined as $\text{outflow}_{\boldsymbol{f}}(v) = -\text{inflow}_{\boldsymbol{f}}(v)$. For a subset $U \subseteq V$, we denote the subset of nodes of U with positive inflow by $U^+_{\boldsymbol{f}}$ (i.e., $U^+_{\boldsymbol{f}} = \{v \in U : \text{inflow}_{\boldsymbol{f}}(v) > 0\}$). Similarly, $U^-_{\boldsymbol{f}} = \{v \in U : \text{inflow}_{\boldsymbol{f}}(v) < 0\}$. We will omit the flow assignment $\boldsymbol{f}$ when it is clear from the context. For example, V^+ is the set of vertices in the entire graph whose inflow is positive.

A flow assignment $\boldsymbol{f}(\cdot)$ is said to *obey conservation* at node v if $\text{inflow}_{\boldsymbol{f}}(v) = 0$. A *feasible flow* is a pseudoflow that obeys conservation at every node that is neither a source nor a sink. That is, $V^+ \subseteq \{S \cup T\}$ and $V^- \subseteq \{S \cup T\}$. A *feasible circulation* is a pseudoflow that obeys conservation at all nodes. That is, $V^+ = V^- = \emptyset$ holds for any feasible circulation. The *value* of a feasible flow $\boldsymbol{f}(\cdot)$ is the sum of the inflow at all sinks, $\sum_{t \in T} \text{inflow}_{\boldsymbol{f}}(t)$ or, equivalently, the sum of outflow at the sources. The maximum flow problem is that of finding a feasible flow with maximum value.

For two flow assignments $\boldsymbol{f}, \boldsymbol{f}'$, the *addition* $\boldsymbol{f} + \boldsymbol{f}'$ is the flow that assigns $\boldsymbol{f}(d) + \boldsymbol{f}'(d)$ to every dart d. Given a flow assignment $\boldsymbol{f}$ (the current flow) and vertex sets A and B, pushing flow from A to B means adding to $\boldsymbol{f}$ a flow $\boldsymbol{f}'$ whose sources (i.e., nodes with negative inflow in $\boldsymbol{f}'$) are only in A and whose sinks (i.e., nodes with positive inflow in $\boldsymbol{f}'$) are only in B.

1.2. Circulations and dual shortest paths.

Let $\boldsymbol{\phi}$ be a function assigning real values to the faces of G. Let $\boldsymbol{\rho}$ be the flow defined by

$$\boldsymbol{\rho}(d) = \boldsymbol{\phi}(\text{head}_{G^*}(d)) - \boldsymbol{\phi}(\text{tail}_{G^*}(d)) \text{ for all darts } d. \tag{1}$$

[1]In some applications capacities are specified for the arcs, not the darts. These can be translated into capacities on darts by setting $\boldsymbol{c}(d)$ to be $\boldsymbol{c}(a)$ for the dart d with the same orientation as a and setting $\boldsymbol{c}(\text{rev}(d))$ to be 0.

Note that here we used the notation equating darts of G with darts of G^*; the head and tail of a dart d in G^* are the faces to the right and left of the dart d in G, respectively. This notation is used throughout the paper. We say that $\boldsymbol{\phi}$ is a *face potential* that induces $\boldsymbol{\rho}$. The inflow at any vertex v is $\text{inflow}_\rho(v) = \sum_{\text{head}(d)=v} \boldsymbol{\rho}(d) = \sum_{\text{head}(d)=v} \boldsymbol{\phi}(\text{head}_{G^*}(d)) - \boldsymbol{\phi}(\text{tail}_{G^*}(d)) = 0$, where the last equality follows from the fact that the darts whose head is v in G form the boundary of a face in G^*, which is a cycle. Therefore, each term in the summation appears once with positive sign and once with negative sign. This shows that a flow assignment induced by face potentials obeys conservation at all nodes.

Let $\boldsymbol{c}$ be a capacity assignment in G. Let x be an arbitrary vertex of G^*. Define the potential $\boldsymbol{\phi}(y)$ of every vertex y of G^* to be the x-to-y distance in G^* (using the capacities $\boldsymbol{c}$ as dart lengths). It is well known (cf. [38]) that the flow $\boldsymbol{\rho}$ induced by the potential $\boldsymbol{\phi}$ is a feasible circulation. This relation between feasible circulations and dual shortest paths is crucially used in our algorithm.

1.3. Residual graphs and invariance of saturation.

A *residual path* in G is a path whose darts all have strictly positive capacities. For two sets of nodes A, B, $A \overset{G}{\to} B$ is used to denote the existence of some residual a-to-b path in G for some nodes $a \in A$ and $b \in B$. Conversely, $A \overset{G}{\not\to} B$ is used to denote that no such path exists. The absence of residual paths is also called *saturation.* We will omit the graph G when it is clear from the context.

The *residual graph* of G with respect to a flow assignment $\boldsymbol{f}(\cdot)$ is the graph $G_{\boldsymbol{f}}$ with the same arc-set, node-set, sources, and sinks and with capacity assignment $\boldsymbol{c_f}(\cdot)$ such that for every dart d, $\boldsymbol{c_f}(d) = \boldsymbol{c}(d) - \boldsymbol{f}(d)$. The operation of *pushing* a flow $\boldsymbol{f'}$ in graph $G_{\boldsymbol{f}}$ yields the graph $G_{\boldsymbol{f}+\boldsymbol{f'}}$.

It is well known that a feasible flow $\boldsymbol{f}$ in G is maximum if and only if $S \overset{G_{\boldsymbol{f}}}{\not\to} T$. We will use several invariants of saturation. Informally these invariants are (1) a circulation does not create any new residual paths; (2) if no augmenting path exists from A to B, then a flow with sources *only* in A cannot create augmenting paths from A to B; and (3) if no augmenting path exists from A to B, then a flow with sinks *only* in B cannot create augmenting paths from A to B. These invariants are formalized by the following lemmas.

LEMMA 1.2. *Let $\boldsymbol{\rho}$ be a circulation. Let u and v be nodes in a graph G. Then*

$$u \overset{G}{\not\to} v \Rightarrow u \overset{G_\rho}{\not\to} v.$$

Proof. Pushing a circulation does not change the amount of flow crossing any cut. This implies that if there was no u-to-v residual path before $\boldsymbol{\rho}$ was pushed, then there is none after $\boldsymbol{\rho}$ is pushed. ∎

LEMMA 1.3 (sources lemma). *Let $\boldsymbol{f}$ be a flow with source set X. Let A, B be two disjoint sets of nodes. Then*

$$A \cup X \overset{G}{\not\to} B \Rightarrow A \overset{G_{\boldsymbol{f}}}{\not\to} B.$$

Proof. The flow $\boldsymbol{f}$ may be decomposed into a cyclic component (a circulation) and an acyclic component. By Lemma 1.2, it suffices to show the lemma for an acyclic flow $\boldsymbol{f}$.

Suppose for the sake of contradiction that there exists a residual a-to-b simple path P after $\boldsymbol{f}$ is pushed for some $a \in A$ and $b \in B$. Let P' be the maximal suffix of

P that was residual before the push. Maximality implies that the dart d of P whose head is start(P') was nonresidual before the push, and $\boldsymbol{f}(\text{rev}(d)) > 0$. The fact that $\boldsymbol{f}(\text{rev}(d)) > 0$ implies that before $\boldsymbol{f}$ was pushed there was a residual path Q from some node $x \in X$ to head(d). Therefore, the concatenation of Q and P' was a x-to-b residual path before the push, a contradiction. ∎

LEMMA 1.4 (sinks lemma). *Let $\boldsymbol{f}$ be a flow with sink set X. Let A, B be two disjoint sets of nodes. Then*

$$A \overset{G}{\not\rightsquigarrow} B \cup X \Rightarrow A \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} B.$$

Proof. The proof is symmetric to the proof of the sources lemma. Again, by Lemma 1.2, it suffices to show the lemma for an acyclic flow $\boldsymbol{f}$. Suppose, for the sake of contradiction, that there exists a residual a-to-b path P after $\boldsymbol{f}$ is pushed for some $a \in A$ and $b \in B$. Let P' be the maximal prefix of P that was residual before the push. That is, the dart d of P whose tail is end(P') was nonresidual before the push, and $\boldsymbol{f}(\text{rev}(d)) > 0$. The fact that $\boldsymbol{f}(\text{rev}(d)) > 0$ implies that before $\boldsymbol{f}$ was pushed there was a residual path Q from tail(d) to some node $x \in X$. Therefore, the concatenation of P' and Q was a a-to-X residual path before the push, a contradiction. ∎

For node sets W, Y, Z, we will use the notation *sources-lemma*$(Z, W \not\rightsquigarrow Y)$ to indicate the use of the sources lemma to establish that if there are no W-to-Y and no Z-to-Y residual paths, then there are no W-to-Y residual paths after a flow with source set Z is pushed. We will use *sinks-lemma*$(W \not\rightsquigarrow Y, Z)$ in a similar manner.

1.4. Converting a pseudoflow into a maximum feasible flow. Let $\boldsymbol{f}$ be a pseudoflow in a graph G with node set V, sources S, and sinks T. We say that a pseudoflow is maximum when $S \cup V^+ \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} T \cup V^-$. A maximum pseudoflow can be converted into a maximum flow in $O(m \log n)$-time for a graph with n nodes and m edges [26, 43]. If G is planar, this can be done in linear time by first canceling flow cycles using the technique of Kaplan and Nussbaum [28] and then by sending back flow from V^+ to S and into V^- from T in topological sort order. We give the details here.

First, the algorithm converts $\boldsymbol{f}$ to an acyclic pseudoflow in linear time [28]. That is, after the conversion there is no cycle C such that $\boldsymbol{f}(d) > 0$ for every dart d of C. Since $\boldsymbol{f}$ is acyclic, the darts with a positive flow induce a topological ordering on the nodes of the graph G. Let v be the last member of V^+ in the topological ordering, and let d be an arbitrary dart that carries flow into v. We set $\boldsymbol{f}(d)$ to $\max\{\boldsymbol{f}(d) - \text{inflow}_{\boldsymbol{f}}(v), 0\}$, and update $\boldsymbol{f}(\text{rev}(d))$ accordingly. The flow assignment $\boldsymbol{f}$ maintains the invariant $S \cup V^+ \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} T \cup V^-$ by *sources-lemma*$(\{v\}, S \cup V^+ \not\rightsquigarrow T \cup V^-)$. Note that the vertex tail(d), which may now belong to V^+ has no residual paths to $T \cup V^-$ since that would imply v had a residual path to $T \cup V^-$. As long as v is in V^+, there must be a dart d which carries flow to v. By changing the flow on d we cannot add to V^+ a new node that appears later than v in the topological ordering. We repeat this process until V^+ is empty. Since we reduce the flow on each dart at most once, this takes linear time. Next we handle V^- while keeping the invariant $S \cup V^+ \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} T \cup V^-$ in a symmetric way, by repeatedly fixing the first vertex of V^- in the topological ordering.

The total running time is $O(n)$, and since V^+, V^- are both empty, we get from the invariant $S \cup V^+ \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} T \cup V^-$ that the resulting pseudoflow is a maximum flow.

1.5. Cycle separators for embedded planar graphs. Miller [37] gave a linear-time algorithm that, given a triangulated two-connected n-node planar embedded graph G with nonnegative node weights, finds a simple cycle, called a cycle separator, consisting of at most $2\sqrt{2}\sqrt{n}$ nodes, such that at most $2/3$ of the total weight is strictly enclosed by the cycle, and at most $2/3$ of the total weight is not enclosed. Short balanced separators are useful as the basis of divide and conquer algorithms, such as the algorithm described in the current paper. The basic idea is to split the original problem into two subproblems in the interior and exterior of the cycle separator. These subproblems are solved recursively. Since the separator is balanced, the depth of the recursion is $O(\log n)$. The fact that the separator is short (i.e., consists of $O(\sqrt{n})$ nodes) is important for being able to efficiently combine the recursive solutions into a solution to the original problem. Cycle separators are particularly useful because the interaction between the solutions obtained recursively is structured along a single simple cycle. This can be exploited to obtain faster algorithms.

1.6. Overview of the algorithm. Consider the following naïve recursive approach for finding a maximum multiple-source multiple-sink flow: split the input graph G in two using a simple cycle separator C and recursively solve the maximum-flow instances induced by each of the two subgraphs. After the two recursive calls have been executed, in each of the two subgraphs there is no residual path from any source to any sink. However, there may still exist residual paths from sources in one subgraph to sinks in the other subgraph. If the algorithm further ensures that in each of the two subgraphs there is no residual path from any source to C and from C to any sink, then, since C is a separator, there is no residual path from any source to any sink in G. The caveat is that a flow assignment that saturates all such residual paths might not be feasible; to saturate all such residual paths, the vertices of the cycle C might have nonzero net inflow/outflow.

The algorithm must therefore solve the following, slightly more general problem in each of the two subgraphs: it must find a pseudoflow such that there is no residual path from a source to a sink or to C and no residual path from C to a sink. We emphasize again that the pseudoflow is not necessarily a feasible flow since the vertices of the cycle C are allowed to have nonzero net inflow/outflow. The formulation of the problem solved by the algorithm needs, in fact, to be even more general. This is because deeper recursive calls must take into account not only the cycle separator C in the current recursive call but also cycle separators of earlier recursive calls. For this overview of the algorithm we ignore this complication. The recursive approach is described in detail in section 2.

After the two recursive calls, there is no residual path from any source to any sink in G. However, the pseudoflow computed by the two recursive calls is not a maximum pseudoflow since there might be residual paths from vertices with positive inflow to vertices with positive outflow. The only such vertices are those of the cycle separator C. The algorithm uses a procedure BalancePath that adjusts the pseudoflow so as to ensure that there is no residual path in G from a vertex of with positive inflow to a vertex with positive outflow. Once this procedure completes its work, the pseudoflow can be turned into a maximum flow in linear time, as described in section 1.4.

The core of the algorithm is the procedure BalancePath. This procedure involves a series of $|C| - 1$ max-flow computations in G. Since $|C|$ is $O(\sqrt{n})$, the challenge is carrying out all these max-flow computations in near-linear time. One of the main novel ideas in our work is that the procedure uses a succinct representation to keep track of the changes to the pseudoflow. The succinct representation does not explicitly store changes to those arcs of the graph not in C but is nevertheless able to carry out augmentations. The basis for this representation is the relation between primal circulations and dual shortest paths described in section 1.2. The procedure also makes use of an adaptation of Fakcharoenphol and Rao's efficient implementation of Dijkstra's algorithm [11]. The representation and procedure are described in section 3.

2. Flow by divide and conquer.

In this section we describe the divide-and-conquer algorithm MSMSMaxFlow in terms of three subprocedures. We will show that the algorithm is correct assuming the correctness of the subprocedures. As we mentioned in the overview in section 1.6, the problem solved by the recursive algorithm is more general than the multiple-source multiple-sink max flow problem. In addition to the sets S and T of sources and sinks, the algorithm is given as input a set A of vertices and is required to produce a pseudoflow that obeys conservation everywhere except at vertices of S, T, or A and such that there are no residual paths from any source to any sink, from any source to any vertex of A, and from any vertex of A to any sink. Each vertex in the set A is the result of contracting a cycle separator used in an earlier recursive call.[2] To solve the maximum flow problem with multiple sources and sinks, we apply algorithm MSMSMaxFlow with $A = \emptyset$.

Algorithm 1 MSMSMaxFlow($G, \boldsymbol{c}, S, T, A$)

Input: a directed planar graph G with nonnegative capacities $\boldsymbol{c}$, a set S of source nodes, a set T of sink nodes, a set $A = \{a_1, \dots, a_k\}$ of size at most six.

Output: a pseudoflow $\boldsymbol{f}$ obeying conservation everywhere except $S \cup T \cup A$, such that $S \overset{G_{\boldsymbol{f}}}{\not\to} T$, $S \overset{G_{\boldsymbol{f}}}{\not\to} A$, $A \overset{G_{\boldsymbol{f}}}{\not\to} T$.

```
1: add zero-capacity arcs to triangulate and two-connect $G$ (required for simple cycle separators)
2: find a balanced (alternately w.r.t. $|G|$ and $|A|$) cycle separator $C$ in $G$
3: ensure that $C$ contains no sources, sinks, or nodes of $A$
4: $P \leftarrow C - \{e\}$, where $e$ is an arbitrary edge of $C$
5: contract all the edges of $P$, turning $e$ into a self loop incident to the only remaining node $v$ of $C$
6: let $G_1$ and $G_2$ be the subgraphs of $G$ strictly enclosed and not enclosed by $e$, respectively
7: $\boldsymbol{f} \leftarrow$ MSMSMaxFlow$(G_1, \boldsymbol{c}|G_1, S \cap G_1, T \cap G_1, (A \cap G_1) \cup \{v\})$
8: $\boldsymbol{f} \leftarrow \boldsymbol{f} +$ MSMSMaxFlow$(G_2, \boldsymbol{c}|G_2, S \cap G_2, T \cap G_2, (A \cap G_2) \cup \{v\})$
9: uncontract the edges of $P$
10: $\boldsymbol{f} \leftarrow$ BalancePath$(G, P, \boldsymbol{c}, \boldsymbol{f})$
11: for $i = 1, 2, \dots, |A|$
12:     $\boldsymbol{f} \leftarrow$ FromCycleLimitedMaxFlow$(G, \boldsymbol{c}, \boldsymbol{f}, C, a_i)$
13:     $\boldsymbol{f} \leftarrow$ ToCycleLimitedMaxFlow$(G, \boldsymbol{c}, \boldsymbol{f}, a_i, C)$
14: (un)push positive excess from $C^+$ and negative excess to $C^-$
15: return $\boldsymbol{f}$
```

The graph is divided using an $O(\sqrt{n})$-size cycle separator C. If C contains a source u, we introduce an artificial node u' and artificial arc $u'u$ with high capacity and designate u' a source instead of u (line 3). Clearly, we can embedded u' and the

[2]It is possible to devise a similar algorithm with a simpler recursive structure. Its running time, however, is slower by a polylogarithmic factor. See [40] for details.

arc $u'u$ without violating the planarity of the graph. Sinks and nodes of A on C are handled similarly. This allows us to assume, for simplicity of presentation, that no nodes of C are sources, sinks, or nodes of A.

All but one edge e of the cycle C are contracted. Thus, the only remaining edge of C is the self loop e, and the only remaining node of C is the node v incident to e.[3] The node v is added to the set A so that, in the recursive calls, v represents the cycle C. In order to bound the running time, we maintain that $|A| \leq 6$. We do so by balancing the separator with respect to $|A|$ every other recursive call (line 2). Since each recursive call increases the size of A by at most 1, and every two recursive calls $|A|$ decreases by a factor of 2/3, the algorithm ensures that $|A| < 6$ at all times. This is similar to previous uses of cycle separators (for example, Fakcharoenphol and Rao [11] or Klein and Subramanian [33]), where the number of *holes* introduced by the separating cycles is bounded.

The algorithm then recurses on the subgraph embedded on each side of the self-loop formed by e. After the recursive calls, the algorithm uncontracts the edges of C. We will show that, at this stage, there are no residual paths from sources to sinks, from sources to nodes of A, and from nodes of A to sinks (Lemma 2.2). However, the flow is not a valid output for MSMSMaxFlow since flow may not be conserved at the nodes of C. We cannot simply push flow back from nodes of C with excess inflow to the sources and from the sinks to nodes of C with excess outflow. There are three reasons this does not work. First, if there is a residual path from a node of C with excess inflow to a node of C with excess outflow, doing so will create a source-to-sink residual path. Second, if there is a residual path from a node of C with excess inflow to a node of A, doing so will create a source-to-A residual path. Third, if there is a residual path from a node of A to a node of C with excess outflow, doing so will create an A-to-sink residual path.

The first problem is corrected by saturating residual paths from nodes of C with excess inflow to nodes of C with excess outflow. This is the most challenging problem of the three, because correcting it requires computing a flow with multiple sources (nodes of C with excess inflow) and multiple sinks (nodes of C with excess outflow). We cannot afford to achieve this by an additional recursive call, because then the running time will no longer be nearly linear. Saturating these residual paths is done by the procedure BalancePath. This procedure is given in section 3; the interface is as follows:

BalancePath($G, P, \boldsymbol{c}, \boldsymbol{f}_0$):
Input a directed planar graph G, simple path P, capacity function $\boldsymbol{c}$, and a pseudoflow $\boldsymbol{f}_0$
Output a pseudoflow $\boldsymbol{f}$ such that
(i) $\boldsymbol{f} - \boldsymbol{f}_0$ satisfies conservation everywhere but P
(ii) $P^+_{\boldsymbol{f}} \overset{G_{\boldsymbol{f}}}{\not\rightsquigarrow} P^-_{\boldsymbol{f}}$
Run time $O(n \log n + |P|^2 \log^2 n)$

The second problem is corrected by saturating residual paths from nodes of C with excess inflow to the nodes of A. Since $|A| < 6$, we can afford to handle each node a_i of A separately. This is done by iterating over the nodes a_i of A and using the procedure FromCycleLimitedMaxFlow to push as much excess flow as possible from C to a_i. This is still challenging because, although the procedure involves only

[3]The remaining self loop has no particular significance. We do not contract e to avoid discussion of contractions of self loops and complications thereof.

a single sink (the node a_i), it involves multiple sources (nodes of C). The details of this procedure are given in section 4 and the interface is as follows:

FROMCYCLELIMITEDMAXFLOW($G, \boldsymbol{c}, \boldsymbol{f}_0, C, t$):

Input a directed planar graph G with capacities $\boldsymbol{c}$, a pseudoflow $\boldsymbol{f}_0$, a simple cycle C, a sink t

Assumes no node of $C^{\oplus}$ has negative inflow w.r.t. $\boldsymbol{f}_0$, where $C^{\oplus}$ is the set of nodes of C reachable from $C^+_{\boldsymbol{f}_0}$ via residual paths ($C^{\oplus} = \{v \in C : C^+_{\boldsymbol{f}_0} \overset{G_{\boldsymbol{f}_0}}{\to} v\}$)

Output a pseudoflow $\boldsymbol{f}$ s.t.

(i) $\boldsymbol{f} - \boldsymbol{f}_0$ obeys conservation everywhere but $C^{\oplus} \cup \{t\}$

(ii) no node of $C^{\oplus}$ has negative inflow w.r.t. $\boldsymbol{f}$

(iii) $C^+_{\boldsymbol{f}} \overset{G_{\boldsymbol{f}}}{\not\to} t$

Run time $O(n \log n + |C|^2 \log^2 n)$.

A similar procedure TOCYCLELIMITEDMAXFLOW is called to push flow from a_i to C in order to correct the third problem.

Next the algorithm pushes back flow from any nodes of C with positive inflow to $S \cup A$ and pushes flow back from $T \cup A$ into any nodes of C with negative inflow (line 14) using the technique described in section 1.4. This ensures that the only imbalances are at S, T, and A.

Running time. The number of nodes of the separator cycle C used to partition G into G_1 and G_2 is $O(\sqrt{|G|})$. Therefore, each invocation of BALANCEPATH, FROMCYCLELIMITEDMAXFLOW, and TOCYCLELIMITEDMAXFLOW in G takes $O(|G| \log |G| + |C|^2 \log^2 |G|) = O(|G| \log^2 |G|)$ time. A standard analysis of the recurrence (e.g., [11]) shows that the algorithm runs in $O(n \log^3 n)$ time.

2.1. Correctness of MSMSMAXFLOW.

We will show that at the end of the MSMSMAXFLOW we have a maximum pseudoflow. More specifically, we will show the following.

THEOREM 2.1. *Upon termination of* MSMSMAXFLOW*:*

(i) $\boldsymbol{f}$ *is a pseudoflow;*

(ii) $\boldsymbol{f}$ *obeys conservation everywhere except at* S, T, *and* A*;*

(iii) $S \overset{G_{\boldsymbol{f}}}{\not\to} T, S \overset{G_{\boldsymbol{f}}}{\not\to} A, A \overset{G_{\boldsymbol{f}}}{\not\to} T$.

The remainder of this section will prove this theorem. We assume for this purpose that the subprocedures are correct. It is easy to see that $\boldsymbol{f}$ is a pseudoflow at every stage in the algorithm as capacity constraints are never violated. Also, $\boldsymbol{f}$ obeys conservation everywhere except at S, T, and A and C. We therefore only need to argue that conservation will be attained at C and that maximality is achieved.

LEMMA 2.2. *At any time in the running of the algorithm after the execution of line* 8 *and before the execution of line* 14, $S \not\to T \cup A \cup C$, *and* $A \cup C \cup S \not\to T$.

Proof. The properties hold just after line 8 by the properties of the recursive calls, by induction on the size of the graph, and by the fact that any residual path between G_1 and G_2 would consist of a residual path to C and a residual path from C. Note that, because of the contractions in line 5, at this time the cycle C consists of just the node v. The nonexistence of residual paths w.r.t. v in the recursive calls implies the nonexistence of residual paths w.r.t. any node of C after the contractions are undone in line 9.

Therefore, just after line 9, there is a cut separating S from $T \cup A \cup C$ that is saturated (i.e., $S \not\to T \cup A \cup C$). BALANCEPATH only pushes flow between vertices of C

and FromCycleLimitedMaxFlow, and ToCycleLimitedMaxFlow only pushes flow between the nodes of A and C. It follows by *sinks-lemma*$(S \nrightarrow T \cup A \cup C, C \cup A)$ that $S \nrightarrow T \cup A \cup C$ at any point after line 8 and before line 14.

A similar argument shows $A \cup C \cup S \nrightarrow T$, giving the lemma. ∎

Let $\boldsymbol{f}_i$ denote the flow assignment at the beginning of iteration i of the loop in line 11. Let $C_i^\oplus$ be the set of nodes of C that are reachable via residual paths from some node of C with positive inflow at the beginning of iteration i of the loop in line 11, i.e., $C_i^\oplus = \{v \in C : C^+ \overset{G_{\boldsymbol{f}_i}}{\rightarrow} v\}$. Likewise define $C_i^\ominus$ as the set of nodes of C that have residual paths to some node of C with negative excess at that time (i.e., $C_i^\ominus = \{v \in C : v \overset{G_{\boldsymbol{f}_i}}{\rightarrow} C^-\}$). Note that, since a node has a (trivial) residual path to itself, $C^+ \subseteq C^\oplus$ with respect to any flow. We will show that after this loop has terminated, we will have $C^+ \nrightarrow C^-$, $C^+ \nrightarrow A$, and $A \nrightarrow C^-$. Combined with Lemma 2.2, and pushing back in/outflow excesses, we will get Theorem 2.1.

We will show that $C^+ \nrightarrow C^-$, $C^+ \nrightarrow A$, and $A \nrightarrow C^-$, by proving this property (iteratively) for $C_i^\oplus$ and $C_i^\ominus$. In order to do so, we will need to relate $C_i^\oplus$ to $C_j^\oplus$ for $i \neq j$.

Lemma 2.3. *For all $i < j$, $C_j^\oplus \subseteq C_i^\oplus$, and $C_j^\ominus \subseteq C_i^\ominus$.*

Proof. It suffices to show that, for all i, $C_{i+1}^\oplus \subseteq C_i^\oplus$ and $C_{i+1}^\ominus \subseteq C_i^\ominus$. We show the former. The proof of the analogous claim for $C_{i+1}^\ominus$ is similar.

Since, by their definition, neither FromCycleLimitedMaxFlow nor ToCycleLimitedMaxFlow create new vertices with inflow excess, $C_{i+1}^+ \subseteq C_i^+$. It therefore suffices to show that no new nodes become reachable from C_i^+ during the execution of lines 12 and 13.

The flow pushed in iteration i of line 12 has sources only in $C_i^\oplus$. Therefore, by *sources-lemma*$(C_i^\oplus, C_i^\oplus \nrightarrow V(C) - C_i^\oplus)$, immediately after line 12 $C_i^\oplus \nrightarrow V(C) - C_i^\oplus$ is maintained. The flow pushed in iteration i of line 13 has sinks only in $C_i^\ominus \subseteq V(C) - C_i^\oplus$. Therefore, by *sinks-lemma*$(C_i^\oplus \nrightarrow V(C) - C_i^\oplus, C_i^\ominus)$, $C_i^\oplus \nrightarrow V - C_i^\oplus$ is maintained immediately after line 13 is executed. ∎

Lemma 2.4. *Just after line 12 in iteration i, $C_i^\oplus \nrightarrow C_i^\ominus$, $C_{i+1}^\oplus \nrightarrow \{a_1, \dots, a_i\}$, $\{a_1, \dots, a_{i-1}\} \nrightarrow C_i^\ominus$, and just after line 13 in iteration i, $C_i^\oplus \nrightarrow C_i^\ominus$, $C_{i+1}^\oplus \nrightarrow \{a_1, \dots, a_i\}$, $\{a_1, \dots, a_i\} \nrightarrow C_{i+1}^\ominus$.*

Proof. Since BalancePath guarantees that $C^+ \nrightarrow C^-$, we have that just after line 10, $C_1^\oplus \nrightarrow C_1^\ominus$, for otherwise there would be a residual path from C^+ to $C_1^\oplus$ to $C_1^\ominus$ to C^- (a contradiction).

The flow pushed in line 12 has sources only in $C_i^\oplus$ so just after line 12

- $C_i^\oplus \nrightarrow C_i^\ominus$ by *sources-lemma*$(C_i^\oplus, C_i^\oplus \nrightarrow C_i^\ominus)$,
- $C_{i+1}^\oplus \nrightarrow a_i$ since $C_{i+1}^\oplus \subseteq C_i^\oplus$ (Lemma 2.3), and by definition of FromCycle-LimitedMaxFlow,
- $C_{i+1}^\oplus \nrightarrow \{a_1, \dots, a_{i-1}\}$ since $C_{i+1}^\oplus \subseteq C_i^\oplus$ (Lemma 2.3), $C_i^\oplus \nrightarrow \{a_1, \dots, a_{i-1}\}$ (inductively), and by *sources-lemma*$(C_i^\oplus, C_i^\oplus \nrightarrow \{a_1, \dots, a_{i-1}\})$,
- $\{a_1, \dots, a_{i-1}\} \nrightarrow C_i^\ominus$ inductively and by *sources-lemma*$(C_i^\oplus, \{a_1, \dots, a_{i-1}\} \nrightarrow C_i^\ominus)$.

Likewise, the flow pushed in line 13 has sinks only in $C_i^\ominus$, so

- $C_i^\oplus \nrightarrow C_i^\ominus$ by *sinks-lemma*$(C_i^\oplus \nrightarrow C_i^\ominus, C_i^\ominus)$,
- $C_{i+1}^\oplus \nrightarrow \{a_1, \dots, a_i\}$ by *sinks-lemma*$(C_{i+1}^\oplus \nrightarrow \{a_1, \dots, a_i\}, C_i^\ominus)$,
- $a_i \nrightarrow C_{i+1}^\ominus$ by definition of ToCycleLimitedMaxFlow,

- $\{a_1,\ldots,a_{i-1}\} \not\rightarrow C^{\ominus}_{i+1}$ since $C^{\ominus}_{i+1} \subseteq C^{\ominus}_i$ (Lemma 2.3), $\{a_1,\ldots,a_{i-1}\} \not\rightarrow C^{\ominus}_i$ (by what is true after line 12), and by *sinks-lemma*$(\{a_1,\ldots,a_{i-1}\} \not\rightarrow C^{\ominus}_i, C^{\ominus}_i)$. ◻

Proof of Theorem 2.1. Since every addition to $\boldsymbol{f}$ during the algorithm respects capacities of all darts, $\boldsymbol{f}$ is a pseudoflow at all times. Inductively, the only nodes that do not obey conservations after the recursive calls are those of S, T, and A. Subsequent changes to $\boldsymbol{f}$ only violate conservation on the nodes of C and A. Therefore, just before line 14, conservation is only violated at S, T, A, and C.

By Lemma 2.2 $S \not\rightarrow C$, and $C \not\rightarrow T$ just before line 14 is executed. By Lemma 2.4 with $i = |A|$ we have $C^{\oplus}_{|A|} \not\rightarrow C^{\ominus}_{|A|}$, $C^{\oplus}_{|A|+1} \not\rightarrow A$, and $A \not\rightarrow C^{\ominus}_{|A|+1}$ just before line 14 is executed. Since the set C^+ (C^-) of nodes of C with positive (negative) inflow just before line 14 is executed is a subset of $C^{\oplus}_{|A|+1}$ ($C^{\ominus}_{|A|+1}$), and since, by Lemma 2.3, $C^{\oplus}_{|A|+1} \subseteq C^{\oplus}_{|A|}$, we have $C^+ \not\rightarrow C^-$, $C^+ \not\rightarrow A$, and $A \not\rightarrow C^-$.

We have shown that $C^+ \not\rightarrow T \cup A \cup C^-$ just before line 14 is executed. Therefore, the flow pushed back from C^+ in line 14 must be pushed to S. Similarly, we have shown that $C^+ \cup S \cup A \not\rightarrow C^-$, so the flow pushed to C^- must be pushed from T.

Let f^+ and f^- be the flow pushed back from C^+ and the flow pushed back into C^-, respectively, in line 14. Just before line 14 is executed we have $S \not\rightarrow A \cup T \cup C$ and $A \cup C \not\rightarrow T$ (by Lemma 2.2) and $C^+ \not\rightarrow C^- \cup A$ and $A \not\rightarrow C^-$ (by Lemma 2.4). Consider first the effect of f^+:

- $S \not\rightarrow A \cup T \cup C^-$ by *sources-lemma*$(C^+, S \not\rightarrow A \cup T \cup C^-)$,
- $A \not\rightarrow T \cup C^-$ by *sources-lemma*$(C^+, A \not\rightarrow T \cup C^-)$.

Next consider the effect of f^-:

- $S \not\rightarrow T \cup A$ by *sinks-lemma*$(S \not\rightarrow T \cup A, C^-)$,
- $A \not\rightarrow T$ by *sinks-lemma*$(A \not\rightarrow T, C^-)$.

This shows that upon termination, $S \not\rightarrow T \cup A$ and $A \not\rightarrow T$. Since any excess inflow or outflow at nodes of C is eliminated when line 14 is executed, the pseudoflow upon termination satisfies conservation everywhere except S, T, A. ◻

3. Balancing the flow on a path.

In this section we present an efficient implementation of the balancing procedure which, roughly speaking, given a path P with nodes having positive, negative, or zero inflow, pushes excess flow between the nodes of the path so that eventually there are no residual paths from nodes of P with positive inflow to nodes of P with negative inflow. We emphasize that the procedure pushes flow in the entire graph but that this flow only violates conservation at the nodes of the path P. Put in other words, the only nodes whose excess flow changes are those of the path P. We begin by describing an abstract algorithm for the balancing procedure and argue its correctness (section 3.1). We then present an *inefficient* implementation of this procedure using Hassin's method for computing st-planar flows (section 3.2); this will introduce key ideas for our *efficient* balancing procedure (section 3.3).

Our balancing procedure resembles a technique used by Johnson and Venkatesan [26]. Let M be the sum of capacities of all of the darts of G. The algorithm first increases the capacities of the darts of the path P and their reverses by M and processes the nodes of P in order $p_1, p_2, \ldots, p_{k+1}$. In iteration i, to processes p_i:

- *Restore the capacities of the darts p_ip_{i+1} and $p_{i+1}p_i$*: Reduce the capacities of p_ip_{i+1} and $p_{i+1}p_i$ by M (i.e., return their capacities back to their original capacities) and change the flow on these darts such that their capacities are respected (i.e., if the flow on these darts exceeds their original capacity, reduce the flow; this changes the flow excess at p_i and p_{i+1}).

- *Eliminate as much excess flow at p_i as possible*: Balance the flow excess at p_i as much as possible by pushing flow (in the entire graph) from p_{i+1} to p_i (if $\text{inflow}(p_i) < 0$) or from p_i to p_{i+1} (if $\text{inflow}(p_i) > 0$). Only push up to $|\text{inflow}(p_i)|$ units of flow.

After doing so, one of the following three holds: (i) the flow is conserved at p_i, (ii) $\text{inflow}(p_i) > 0$ and there are no residual paths from p_i to any of the other nodes of P (since the large capacities on darts between unprocessed nodes of P allow flow to be forwarded to p_{i+1}), or (iii) $\text{inflow}(p_i) < 0$ and there are no residual paths to p_i from any of the other nodes of P.

3.1. Correctness of abstract balancing procedure. We now argue that the abstract balancing procedure satisfies the requirements in the definition of BALANCEPATH. Namely, it produces a pseudoflow $\boldsymbol{f}$ such that

(i) $\boldsymbol{f} - \boldsymbol{f}_0$ satisfies conservation everywhere but P,

(ii) $P^+_{\boldsymbol{f}} \overset{G_{\boldsymbol{f}}}{\not\twoheadrightarrow} P^-_{\boldsymbol{f}}$.

Clearly the flow always satisfies conservation everywhere except at nodes of P since the abstract procedure only pushes flow between nodes of P. We show that the procedure achieves the property $P^+ \not\twoheadrightarrow P^-$ inductively. Let P_i be the prefix subpath of P that ends in the ith vertex, p_i.

LEMMA 3.1. *At the end of iteration i of the balancing procedure we have the invariants:*

$$\text{(i)}\;\; P^+_i \not\twoheadrightarrow (P - P_i),\; \text{(ii)}\;\; (P - P_i) \not\twoheadrightarrow P^-_i,\; \textit{and}\; \text{(iii)}\;\; P^+_i \not\twoheadrightarrow P^-_i$$

Proof. This proof is by induction on the iteration number of the loop. For $i = 0$ the invariants are trivially satisfied. We assume the invariants hold at the beginning of the ith iteration (namely, $P^+_{i-1} \not\twoheadrightarrow (P - P_{i-1})$, $(P - P_{i-1}) \not\twoheadrightarrow P^-_{i-1}$, and $P^+_{i-1} \not\twoheadrightarrow P^-_{i-1}$), and we show that they still hold at the end of the ith iteration.

If flow at p_i is conserved at the end of the ith iteration, then $P^+_{i-1} = P^+_i$ and $P^-_{i-1} = P^-_i$. We get the lemma by the inductive hypothesis and by (i) *sinks-lemma*$(P^+_{i-1} \not\twoheadrightarrow (P - P_{i-1}), \{p_i, p_{i+1}\})$, (ii) *sources-lemma*$(\{p_i, p_{i+1}\}, (P - P_{i-1}) \not\twoheadrightarrow P^-_{i-1})$, and (iii) *sinks-lemma*$(P^+_{i-1} \not\twoheadrightarrow P^-_{i-1}, \{p_i, p_{i+1}\})$.

If p_i has positive inflow at the *end* of iteration i, then, since we only push as much flow as there is inflow at p_i at the *start* of iteration i, the flow pushed must have been a maximum flow from p_i to p_{i+1}. Since the capacities of the darts in $P - P_i$ are sufficiently large, the maximality of the flow implies $p_i \not\twoheadrightarrow (P - P_i)$. $P^+_{i-1} \not\twoheadrightarrow (P - P_i)$ is achieved since $P_{i-1} \subset P_i$ and *sinks-lemma*$(P^+_{i-1} \not\twoheadrightarrow (P - P_{i-1}), p_{i+1})$. This shows invariant (i).

Since in this case $P^-_i = P^-_{i-1}$, by the inductive hypothesis ($(P - P_{i-1}) \not\twoheadrightarrow P^-_{i-1}$), and by *sources-lemma*$(p_i, (P - P_{i-1}) \not\twoheadrightarrow P^-_{i-1})$, we get $(P - P_i) \not\twoheadrightarrow P^-_i$, completing the proof of invariant (ii).

For invariant (iii), since $p_i \in (P - P_{i-1})$, we have by the inductive hypothesis $p_i \not\twoheadrightarrow P^-_{i-1}$, as well as $P^+_{i-1} \not\twoheadrightarrow P^-_{i-1}$ at the start of iteration i. These two properties remain true at the end of iteration i by *sources-lemma*$(p_i, p_i \not\twoheadrightarrow P^-_{i-1})$ and *sinks-lemma*$(P^+_{i-1} \not\twoheadrightarrow P^-_{i-1}, p_i)$, respectively. This proves invariant (iii) since in this case $P^+_i = P^+_{i-1} \cup \{p_i\}$ and $P^-_i = P^-_{i-1}$.

The case of negative inflow at p_i is similar. ∎

3.2. An inefficient implementation for pushing flow. In this section, we give an *inefficient* implementation for balancing the flow on a path P. This will facilitate the explanation of the efficient procedure in the next section. The idea is based on Hassin's algorithm for maximum flow in planar graphs between adjacent vertices and shortest path computations in the dual graph.

3.2.1. Hassin's algorithm for maximum *st*-planar flow. A planar graph is st-planar if nodes s and t are incident to the same face. Hassin [19] gave an algorithm for computing a maximum flow from s to t in an st-planar graph. We briefly describe this algorithm here since we will use it to push flow in BalancePath.

Hassin's algorithm starts by embedding in G an artificial, infinite capacity arc a from t to s in a face common to both s and t. It then computes shortest-path distances in G^*: $\boldsymbol{\phi}[f]$ is the distance to face f (of G) from the face (in G) to the right of a, using capacities as dart lengths. Consider the flow:

$$\boldsymbol{\rho}[d] = \boldsymbol{\phi}[\text{head}_{G^*}(d)] - \boldsymbol{\phi}[\text{tail}_{G^*}(d)] \text{ for all darts } d. \tag{2}$$

That is, $\boldsymbol{\rho}$ is the feasible circulation induced by the face potential vector $\boldsymbol{\phi}$. It can be shown that, after removing the artificial arc a from G, $\boldsymbol{\rho}$ is a maximum feasible flow from s to t in G. For our purposes, we require a maximum flow of *value at most* x rather than a maximum flow; it is not difficult to see that setting the capacity of the artificial arc a to x instead of infinity results in the desired limited maximum flow [30].

3.2.2. Using Hassin's algorithm. We use Hassin's algorithm to push flow in each iteration. The change to the flow in iteration i is a flow between the endpoints p_i and p_{i+1} of the ith dart, d_i, of P. This flow can be represented as the sum of a circulation $\boldsymbol{\rho}_i$ and a flow on d_i and $\text{rev}(d_i)$. The former is the circulation in Hassin's algorithm, and the latter represents the adjustment to the circulation that results from removing the artificial arc in Hassin's algorithm. In preparation for giving an efficient implementation, we represent the circulations implicitly by their corresponding face potentials $\boldsymbol{\phi}_i$. By linearity of (2), the sum $\boldsymbol{\phi} = \sum_{j=1}^{i} \boldsymbol{\phi}_j$ is the face potential vector that induces a circulation $\boldsymbol{\rho}$. The final flow on darts not in P is just the flow assigned to them by this circulation. The final flow on darts in P is given by a flow that we maintain explicitly plus this circulation.

Refer to Algorithm 2, InefficientBalancePath. Throughout we maintain an explicit flow assignment $\boldsymbol{f}$ and a potential $\boldsymbol{\phi}$ that represents the circulation $\boldsymbol{\rho}$. The net flow on a dart d, denoted netflow(d), is given by $\boldsymbol{f}[d] + \boldsymbol{\rho}[d] = \boldsymbol{f}[d] + \boldsymbol{\phi}[\text{head}_{G^*}(d)] - \boldsymbol{\phi}[\text{tail}_{G^*}(d)]$. Given the vectors $\boldsymbol{f}$ and $\boldsymbol{\phi}$, netflow(d) can be computed in constant time. The imbalance at a vertex p of P is always computed with respect to netflow; we represent this by inflow(p), which can be computed on demand. However, for the efficient implementation in the next section, we will explicitly maintain an array representing inflow. Updates to an entry of this array can clearly be made in constant time when $\boldsymbol{f}[d]$ for a dart d of P incident to p is updated. Updates to $\boldsymbol{f}[d]$ always take the flow represented by the circulation $\boldsymbol{\rho}[d] = \boldsymbol{\phi}[\text{head}_{G^*}(d)] - \boldsymbol{\phi}[\text{tail}_{G^*}(d)]$ into account (e.g., line 7).

In our use of Hassin's algorithm for pushing flow, instead of using an artificially introduced arc a from t to s, we use an existing dart d from the source to the sink (d is either d_i or $\text{rev}(d_i)$, depending on the value of inflow(p_i)). In order for this to work, the capacity of d must be zero (as is indeed the case for $\text{rev}(a)$ in Hassin's algorithm). We achieve this by either saturating d or pushing $|\text{inflow}(p_i)|$ units of flow and balancing p_i (line 10).

The core of Hassin's algorithm is implemented in lines 14– 15. The flow we find via dual-shortest-path potentials (line 14) is a circulation rather than a maximum flow.

Algorithm 2 InefficientBalancePath

Input: directed planar graph G, simple path P with nodes $p_1, p_2, \ldots, p_{k+1}$ and darts $d_1, d_2, \ldots, d_k$, capacity function $\boldsymbol{c}$ and pseudoflow $\boldsymbol{f}_0$

Output: pseudoflow $\boldsymbol{f}$ s.t. (i) $\boldsymbol{f} - \boldsymbol{f}_0$ satisfies conservation everywhere but P, and (ii) $P^+ \nrightarrow P^-$

```
1: $\boldsymbol{f} \leftarrow \boldsymbol{f}_0$
2: $\boldsymbol{c}[d] \leftarrow \boldsymbol{c}[d] + M$ for all darts $d$ of $P \cup \mathrm{rev}(P)$
3: $\boldsymbol{\phi}[f] \leftarrow 0$ for all faces $f$ of $G$          initialize potentials representing circulation
4: for $i = 1, 2, \ldots, k$
5:     $\boldsymbol{c}[d_i] \leftarrow \boldsymbol{c}[d_i] - M$,  $\boldsymbol{c}[\mathrm{rev}(d_i)] \leftarrow \boldsymbol{c}[\mathrm{rev}(d_i)] - M$ return capacities back to their original
6:     if netflow$(d_i) > \boldsymbol{c}[d_i]$ then
7:         $\boldsymbol{f}[d_i] \leftarrow \boldsymbol{c}[d_i] - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d_i)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d_i)]$ ,  $\boldsymbol{f}[\mathrm{rev}(d_i)] \leftarrow -\boldsymbol{f}[d_i]$ and adjust
           the flow accordingly
8:     if inflow$(p_i) > 0$ then $d \leftarrow d_i$ else $d \leftarrow \mathrm{rev}(d_i)$     determine the direction to push flow
9:     $c = \boldsymbol{c}[d] - \boldsymbol{f}[d] - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d)]$          residual capacity of $d$
10:    $\boldsymbol{f}[d] \leftarrow \boldsymbol{f}[d] + \min(\boldsymbol{c}[d] - \mathrm{netflow}(d), |\mathrm{inflow}(p_i)|)$,  $\boldsymbol{f}[\mathrm{rev}(d)] \leftarrow -\boldsymbol{f}[d]$ balance on
           $d$ as much as possible
11:    if $|\mathrm{inflow}(p_i)| > 0$ then
12:        $\ell[d'] \leftarrow \boldsymbol{c}[d'] - \mathrm{netflow}(d')$ for each dart $d'$ of $G$      lengths are residual capacities
13:        $\ell[\mathrm{rev}(d)] \leftarrow |\mathrm{inflow}(p_i)|$          limit the amount of flow we push in balancing
14:        $\boldsymbol{\phi}_i \leftarrow$ shortest-path distances in $G^*$ from $\mathrm{head}_{G^*}(d)$ with lengths given by $\ell$
15:        $\boldsymbol{f}[d] \leftarrow \boldsymbol{f}[d] + \boldsymbol{\phi}_i[\mathrm{tail}_{G^*}(d)]$,  $\boldsymbol{f}[\mathrm{rev}(d)] \leftarrow -\boldsymbol{f}[(d)]$   compensate for flow $\boldsymbol{\phi}_i$ pushes
           on rev$(d)$
16:        $\boldsymbol{\phi} = \boldsymbol{\phi} + \boldsymbol{\phi}_i$          accumulate the current circulation
17: return netflow
```

In Hassin's algorithm it is converted to a maximum flow by removing the artificial arc a. In our implementation the convertion is done by removing the flow from the dart rev(d) (line 15). Note that we explicitly represent the flow on d_i and rev(d_i) separately from the potentials used to represent the circulations (line 18).

3.3. An efficient implementation of BalancePath.

In this section, we give an *efficient* implementation for the abstract balancing procedure that runs in time $O(n \log n + |P|^2 \log^2 n)$. We first review the necessary tools.

3.3.1. Price functions and reduced lengths.

For a directed graph G with dart-lengths $\ell(\cdot)$, a *price function* is a function ϕ from the nodes of G to the reals. For a dart d, the *reduced length with respect to* ϕ is $\ell_\phi(d) = \ell(d) + \phi(\mathrm{tail}(d)) - \phi(\mathrm{head}(d))$. We say that a price function *reduces* the lengths.

Price functions can be useful for converting a shortest-path problem involving positive and negative lengths to one involving only nonnegative edge lengths [25]. For example, single-source distances reduce lengths to be nonnegative. This can be useful in multiple-source shortest path computations: for an arbitrary node r, let $\phi(v)$ be the r-to-v distance in G with respect to $\ell(\cdot)$; then for every arc uv, $\phi(v) \leq \phi(u) + \ell(uv)$, so $\ell_\phi(uv) \geq 0$. Here we assume that all distances are finite (i.e., that all nodes are reachable from r).[4] Note that shortest paths with respect to ℓ remain shortest with respect to the reduced length ℓ_ϕ. (The length of each path is shifted by the difference

[4]This assumption is w.l.o.g. since we can always add arcs with sufficiently large lengths to make all nodes reachable without affecting the shortest paths in the graph.

of potentials of its endpoints.) Therefore, shortest paths in G can now be computed using Dijkstra's algorithm, rather than by slower methods that can handle negative lengths.

3.3.2. Fakcharoenphol and Rao's Dijkstra implementation. Fakcharoenphol and Rao [11] described a data structure that can be used in a procedure to efficiently implement Dijkstra's algorithm among the vertices of a cycle separator in a planar graph. We adapt this procedure to maintain dual distances between faces adjacent to the path we are balancing; these distances will give us the potential function that defines the set of circulations used in the inefficient implementation. We use a variant of Fakcharoenphol and Rao's procedure that additionally uses a price function that reduces lengths to be nonnegative. Specifically, this data structure gives the following:

FRDIJKSTRA($H, X, \mathrm{D}, P, \ell, v, \phi$):

Input a planar graph H, a subset of nodes X, a table D storing the pairwise distances in H among X, a set of darts P (not necessarily in H and $P \cup H$ not necessarily planar) whose endpoints are nodes in X with lengths ℓ and a node $v \in X$, a price function ϕ for the nodes of X

Assumes the nodes of X are on the boundary of a single face of H and in clockwise order around the boundary of this face, ϕ reduces ℓ and D to be nonnegative

Output the distances in $H \cup P$ from v to every node in X w.r.t. the lengths reduced by ϕ

Run time $O(|X| \log^2 |X| + |P| \log |X|)$

The procedure of Fakcharoenphol and Rao partitions the table D into several subtables $\{\mathrm{D}_\alpha\}_\alpha$ that correspond to distances between pairs of disjoint sets of nodes of X, where each set consists of nodes that are consecutive in X. Fakcharoenphol and Rao use, for each such subtable D_α, a data structure supporting range minimum queries of the form $\min_{j_1 \le j \le j_2}\{\mathrm{D}_\alpha[i,j]\}$ for every i, j_1, j_2 in $O(\log |X|)$ time [11, footnote on p. 884]. Without the price function, this can be achieved using a range-search tree [9] for every row i of D_α. However, a range-search tree would first need to compute the $O(|X|^2)$ reduced lengths of D. Instead, we use Monge range-query data structures, due to Kaplan et al. [27], which can be constructed from the table D in $O(|X| \log |X|)$ time, and answer queries of the desired form in $O(\log |X|)$ time.

3.3.3. The procedure for efficiently balancing a path. Finally, we describe the efficient implementation. Refer to Algorithm 3, BALANCEPATH. As in the inefficient implementation, the procedure will maintain the total flow as the sum of a circulation $\boldsymbol{\rho}$ and a flow assignment $\boldsymbol{f}$ that differs from the input flow $\boldsymbol{f}_0$ only on the darts of $P \cup \mathrm{rev}(P)$. Initially $\boldsymbol{f}$ is set equal to $\boldsymbol{f}_0$. The circulation $\boldsymbol{\rho}$ will be represented by a face-potential vector $\boldsymbol{\phi}$. However, we will show that it suffices to maintain just those entries $\boldsymbol{\phi}^X$ (line 6) of $\boldsymbol{\phi}$ that correspond to faces incident to P. In the pseudocode we highlight the changes from INEFFICIENTBALANCEPATH.

Precomputing distances. We start by computing distances that will be invariant throughout the algorithm in lines 1 to 3. We use dart-lengths inherited from residual capacities. The distances we precompute are distances in the dual between nodes X^* that are endpoints of darts of P (i.e., corresponding to the faces adjacent to P). We compute distances in the graph H^* obtained from G^* by removing the darts of P. Note that deleting P in the dual is the same as contracting the darts of P in the primal. In the primal, this results in a single vertex v and every face that was

Algorithm 3 BalancePath

Input: directed planar graph G, simple path P with nodes $p_1, p_2, \ldots, p_{k+1}$ and darts $d_1, d_2, \ldots, d_k$, capacity function $\boldsymbol{c}$ and pseudoflow $\boldsymbol{f}_0$
Output: pseudoflow $\boldsymbol{f}$ s.t. (i) $\boldsymbol{f} - \boldsymbol{f}_0$ satisfies conservation everywhere but P, and (ii) $P^+ \not\twoheadrightarrow P^-$

```
1:  $X^* \leftarrow$ set of endpoints in the planar dual $G^*$ of the darts of $P$
2:  $H^* \leftarrow$ subgraph of $G^*$ with the darts of $P$ removed
3:  $\mathrm{D}^* \leftarrow$ table of pairwise distances in $H^*$ between nodes of $X^*$ w.r.t.
          the lengths $\boldsymbol{c} - \boldsymbol{f}_0$
4:  $\boldsymbol{f} \leftarrow \boldsymbol{f}_0$
5:  $\boldsymbol{c}[d] \leftarrow \boldsymbol{c}[d] + M$ for all darts $d$ of $P \cup \mathrm{rev}(P)$
6:  $\boldsymbol{\phi}^X[x] \leftarrow 0$ for all nodes $x$ of $X^*$
7:  for $i = 1, 2, \ldots, k$
8:     $\boldsymbol{c}[d_i] \leftarrow \boldsymbol{c}[d_i] - M$,  $\boldsymbol{c}[\mathrm{rev}(d_i)] \leftarrow \boldsymbol{c}[\mathrm{rev}(d_i)] - M$
9:     if netflow$(d_i) > \boldsymbol{c}[d_i]$ then $\boldsymbol{f}[d_i] \leftarrow \boldsymbol{c}[d_i] - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d_i)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d_i)]$ ,
       $\boldsymbol{f}[\mathrm{rev}(d_i)] \leftarrow -\boldsymbol{f}[d_i]$
10:    if inflow$(p_i) > 0$ then $d \leftarrow d_i$ else $d \leftarrow \mathrm{rev}(d_i)$
11:    $c = \boldsymbol{c}[d] - \boldsymbol{f}[d] - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d)]$
12:    $\boldsymbol{f}[d] \leftarrow \boldsymbol{f}[d] + \min(\boldsymbol{c}[d] - \mathrm{netflow}(d), |\mathrm{inflow}(p_i)|)$,  $\boldsymbol{f}[\mathrm{rev}(d)] \leftarrow -\boldsymbol{f}[d]$
13:    if $|\mathrm{inflow}(p_i)| > 0$ then
14:       $\ell[d'] \leftarrow \boldsymbol{c}[d'] - \mathrm{netflow}(d')$ for each dart $d'$ of $P \cup \mathrm{rev}(P)$   lengths of darts of $P$
          are residual capacities
15:       $\ell[\mathrm{rev}(d)] \leftarrow |\mathrm{inflow}(p_i)|$
16:       $\boldsymbol{\phi}_i^X(\cdot) \leftarrow$FRDijkstra$(H^*, X^*, \mathrm{D}^*, P, \ell, \mathrm{head}_{G^*}(d), \boldsymbol{\phi}^X)$   compute the balancing
          circulation
17:       $\boldsymbol{f}[d] \leftarrow \boldsymbol{f}[d] + \boldsymbol{\phi}_i^X[\mathrm{tail}_{G^*}(d)]$,  $\boldsymbol{f}[\mathrm{rev}(d)] \leftarrow -\boldsymbol{f}[(d)]$   compensate for flow $\boldsymbol{\phi}_i^X$ pushes
          on rev$(d)$
18:       $\boldsymbol{\phi}^X \leftarrow \boldsymbol{\phi}^X + \boldsymbol{\phi}_i^X$   accumulate the current circulation
19: $\boldsymbol{\chi} \leftarrow$ shortest-path distances in $G^*$ from any vertex $x$ using lengths $\boldsymbol{c}[d] - \boldsymbol{f}[d]$
20: $\boldsymbol{f}'[d] \leftarrow \boldsymbol{f}[d] + \boldsymbol{\chi}[\mathrm{head}_{G^*}(d)] - \boldsymbol{\chi}[\mathrm{tail}_{G^*}(d)]$ for every dart $d \in G$
21: return $\boldsymbol{f}'$
```

adjacent to P is now adjacent to v (Figure 1). In the dual v is a face and X^* are nodes on the boundary of that face. We get the following.

Observation 3.2. *The nodes X^* are adjacent to a single face in H^*.*

Limiting computation to faces adjacent to P. The main difference between the inefficient and efficient algorithms is that rather than computing the entire dual shortest-path tree to implement Hassin's algorithm (line 14 of InefficientBalanceFlow) we only compute distances to nodes in X^* (line 16 of BalanceFlow). We do so using FRDijkstra, made possible by the precomputed distances D and Observation 3.2. We now argue why this is sufficient.

The flow on a dart d in the primal is

$$\boldsymbol{f}[d] + \boldsymbol{\phi}[\mathrm{head}_{G^*}(d)] - \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d)]. \tag{3}$$

Therefore the residual capacity of d is

$$(\boldsymbol{c}[d] - \boldsymbol{f}[d]) - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d)], \tag{4}$$

which is its *reduced* length $\ell_{\boldsymbol{\phi}}[d]$ with respect to the length $\ell(\cdot)$ and price function $\boldsymbol{\phi}$.

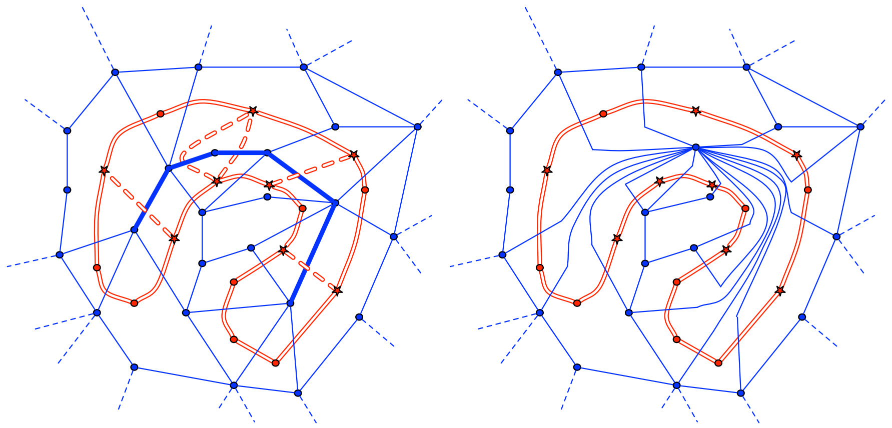

FIG. 1. *The left diagram shows part of the graph G and some edges of its dual G^*. Edges of G are solid blue. Edges of P are bold. Dual edges are double lined red. The dual edges of P are in double lined dashed red. The nodes of X^* are star-shaped. The right diagram shows the situation after deleting the dual edges of P in order to obtain H^*. The nodes of X^* are on the boundary of a single face of H^*.*

Suppose that d belongs to H^*, i.e., d is not one of the darts of $P \cup \mathrm{rev}(P)$. The procedure never changes $\boldsymbol{f}[d]$, so $\boldsymbol{f}[d] = \boldsymbol{f}_0[d]$. Therefore $\ell(d) = \boldsymbol{c}[d] - \boldsymbol{f}_0[d]$. These lengths are known at the beginning of the procedure's execution. The reduced length of an X^*-to-X^* path $Q = d'_1, d'_2, \cdots, d'_j$ in H^* is

(5)
$$\sum_{i=1}^{j} (\ell(d'_i) - \boldsymbol{\phi}[\mathrm{head}_{G^*}(d'_i)] + \boldsymbol{\phi}[\mathrm{tail}_{G^*}(d'_i)]) = \left(\sum_{i=1}^{j} \ell(d'_i)\right) - \boldsymbol{\phi}[\mathrm{end}(Q)] + \boldsymbol{\phi}[\mathrm{start}(Q)].$$

Therefore, for any nodes $x, y \in X^*$, the x-to-y distance in H^* w.r.t. the residual capacity is given by $\mathrm{D}^*[x,y] - \boldsymbol{\phi}[y] + \boldsymbol{\phi}[x]$. Since the procedure maintains the restriction of $\boldsymbol{\phi}$ to nodes of X^*, this distance can be obtained in constant time. The flow of all darts not in H^* (i.e., the darts of P and their reverses) is maintained explicitly, so the residual capacity of each such dart can also be obtained in constant time.

It follows from the above discussion that the potential function $\boldsymbol{\phi}^X$ computed by FRDIJKSTRA in line 16 of the efficient implementation is the restriction to X^* of the potential function $\boldsymbol{\phi}$ that would have been computed by the inefficient implementation. Therefore, the net flow maintained by BALANCEFLOW is the same as that maintained by INEFFICIENTBALANCEFLOW.

In order to output the net flow for all darts in G (not just in P), we need to know the entire potential function $\boldsymbol{\phi}$ rather than just its restriction $\boldsymbol{\phi}^X$. Observe, however, that any pseudoflow that differs from $\boldsymbol{f}$ by a circulation is a valid output of BALANCEPATH since a circulation does not change the inflow at any node, nor does it introduce residual paths between nodes that are not connected by a residual path in $G_{\boldsymbol{f}}$. It therefore suffices to find any feasible circulation $\boldsymbol{\chi}$ in $G_{\boldsymbol{f}}$. This can be done by computing shortest paths in G^* w.r.t. the lengths $\boldsymbol{c} - \boldsymbol{f}$ from an arbitrary node $x \in G^*$ (line 19).

Avoiding negative-lengths shortest paths. Note that while netflow always respects capacities, $\boldsymbol{f}$ may not (as the circulations influence netflow). Therefore the lengths used for the shortest-path computation in line 19 may be negative. We show that it is not necessary to use a generic negative-lengths shortest-path algorithm. While this will not improve the running time of MSMSMaxFlow, it will improve the running time of BalanceFlow. We also think doing so is more elegant. We replace line 19 with

```
1: $\ell[d'] \leftarrow \boldsymbol{c}[d'] - \boldsymbol{f}[d']$ for every $d' \in P \cap \mathrm{rev}(P)$    these lengths may be negative
2: $x \leftarrow$ arbitrary node in $X^*$
3: $\boldsymbol{\psi}^X \leftarrow$ FRDijkstra$(H^*, X^*, \mathrm{D}^*, P, \ell, x, \boldsymbol{\phi}^X)$    $\ell$ reduced by $\phi^X$ are nonnegative
4: $\boldsymbol{\chi}[y] \leftarrow \infty$ for every node $y$ of $G^*$
5: $\boldsymbol{\chi}[y] \leftarrow \boldsymbol{\psi}^X[y] - \boldsymbol{\phi}^X[x] + \boldsymbol{\phi}^X[y]$ for every $y \in X^*$
6: $\ell[d] \leftarrow \boldsymbol{c}[d] - \boldsymbol{f}_0[d]$ for every dart $d \notin P$
7: $\boldsymbol{\chi}[y] \leftarrow$ shortest path distance in $G^*$ from $x$ to $y$ using dart-lengths $\ell$ for every $y \notin X^*$
```

In line 3 we compute distances from x to every node in X^* w.r.t. the lengths ℓ reduced by $\boldsymbol{\phi}^X$, which are nonnegative. By the same argument as in (5), the potentials for faces adjacent to P computed in line 5 are correct. The dart-lengths computed in line 6 correspond to residual capacities with respect to $\boldsymbol{f}$ because $\boldsymbol{f}[d] = \boldsymbol{f}_0[d]$ for every dart $d \notin P$. These lengths are nonnegative since $\boldsymbol{f}_0$ is a pseudoflow. Since the distances to every node in X^* have already been computed and the negative length darts have both endpoints in X^*, we can initialize Dijkstra's algorithm with these values and compute the remaining distances without worrying about negative-length darts.

Running time. Let n be the number of nodes in G.

Since the nodes in X^* are the nodes adjacent to a single face, we can implicitly compute D^* using the multiple-source shortest-path algorithm due to Klein [31] in $O(n \log n)$ time and populate the entries of D^* in $O(k^2 \log n)$ time.

The execution of each iteration is dominated by the call to FRDijkstra, which takes $O(k \log^2 k)$ time. Since there are k iterations, all iterations take a total of $O(k^2 \log^2 k)$ time.

The running time for computing $\boldsymbol{\chi}$ by the method that avoids negative-length shortest path computations takes $O(k \log^2 k + n)$ time, using the linear-time shortest-path algorithm of Henzinger et al. [20]. The total running time of BalanceFlow is therefore $O(n \log n + k^2 \log^2 n)$ as desired.

4. Pushing excess inflow from a cycle.

In this section we describe the procedure FromCycleLimitedMaxFlow that pushes excess inflow from a cycle to a node not on the cycle, Algorithm 4. Since the procedure ToCycleLimitedMaxFlow is very similar, we omit its description.

Recall that $C^{\oplus}$ consists of all nodes of C reachable via residual paths from the nodes of $C^{+}_{\boldsymbol{f}_0}$. Thus, there is an output flow that involves no darts incident to nodes in $C - C^{\oplus}$. It follows that restricting the computation to the graph obtained from G by deleting the nodes in $C - C^{\oplus}$ (line 1) does not restrict the computed flow. After the deletion, introducing darts between consecutive nodes of $C^{\oplus}$ (line 2) will not violate planarity. The idea is to use a single-source single-sink maximum flow algorithm to push the excess flow from $C^{\oplus}$ to t. This is done by contracting $C^{\oplus}$ so that turns into a single node s (line 3). Then in line 4 a maximum st-flow is computed using the single-source single-sink algorithm in [3]. As in Algorithm 1, this feasible flow is a pseudoflow in the uncontracted graph, and may contain residual paths between nodes with positive and negative flow excess. The procedure therefore balances the flow along $C^{\oplus}$ (line 6). Let $\boldsymbol{f}$ be the flow pushed by the procedure up until just

Algorithm 4 FromCycleLimitedMaxFlow$(G, \boldsymbol{c}, \boldsymbol{f}_0, C, t)$

Input: a directed planar graph G with capacities $\boldsymbol{c}$, a pseudoflow $\boldsymbol{f}_0$, a simple cycle C, a sink t.
Assumes: no node of $C^\oplus$ has negative inflow w.r.t. $\boldsymbol{f}_0$, where $C^\oplus$ is the set of nodes of C reachable from $C^+_{\boldsymbol{f}_0}$ via residual paths ($C^\oplus = \{v \in C : C^+_{\boldsymbol{f}_0} \overset{G_{\boldsymbol{f}_0}}{\to} v\}$)
Output: a pseudoflow $\boldsymbol{f}$ s.t. (i) $\boldsymbol{f} - \boldsymbol{f}_0$ obeys conservation everywhere but $C^\oplus \cup \{t\}$, (ii) no node of $C^\oplus$ has negative inflow w.r.t. $\boldsymbol{f}$, and (iii) $C^+_{\boldsymbol{f}} \overset{G_{\boldsymbol{f}}}{\not\to} t$.

```
1: delete the nodes of $C - C^\oplus$
2: introduce 0-capacity darts to create a path $P$ that visits the nodes of $C^\oplus$ according
   to their cyclic order on $C$
3: contract the darts of $P$, collapsing $C^\oplus$ into the single node $s$
4: $\boldsymbol{f} \leftarrow$ maximum $st$-flow w.r.t. capacities $\boldsymbol{c} - \boldsymbol{f}_0$
5: uncontract the edges of $P$
6: $\boldsymbol{f} \leftarrow$BalancePath$(G, P, \boldsymbol{c}, \boldsymbol{f}_0 + \boldsymbol{f}) - \boldsymbol{f}_0$
7: push back flow in $\boldsymbol{f}$ to nodes of $C^\oplus$ whose inflow is negative
8: return $\boldsymbol{f}_0 + \boldsymbol{f}$
```

after line 6. Some nodes of $C^\oplus$ may have negative inflow with respect to $\boldsymbol{f} + \boldsymbol{f}_0$. Since there may not be such nodes with respect to the pseudoflow returned by the procedure, line 7 pushes back excess flow in $\boldsymbol{f}$ from t to these nodes so as to make their inflow zero. This is done using the procedure of section 1.4.

Correctness. Any flow that is pushed by the algorithm originates at $C^\oplus$ and terminates at $C^\oplus \cup \{t\}$. Therefore, $\boldsymbol{f} - \boldsymbol{f}_0$ violates conservation only at $C^\oplus \cup \{t\}$.

After uncontracting P, $\boldsymbol{f}$ is a maximum $C^\oplus$-to-t flow in G w.r.t. the residual capacities $\boldsymbol{c} - \boldsymbol{f}_0$. However, some of the nodes of $C^\oplus$ may now have negative inflow w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$. BalancePath reroutes the flow among the nodes of $C^\oplus$ so that, w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$, there are no residual paths from nodes of $C^\oplus$ with positive inflow to nodes of $C^\oplus$ with negative inflow. This implies that any node of $C^\oplus$ that still has negative inflow has pushed too much flow to t. Line 7 modifies $\boldsymbol{f}$ to push back such excess flow so that no node of $C^\oplus$ has negative inflow w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$. We can do so by only pushing back flow of $\boldsymbol{f}$ (rather than flow of $\boldsymbol{f}_0 + \boldsymbol{f}$) since no node of $C^\oplus$ has negative inflow w.r.t. $\boldsymbol{f}_0$.

By maximality of the flow pushed in line 4, just after line 4 is executed there are no $C^\oplus$-to-t residual paths. Clearly, this remains true when the capacities of the artificial darts are set to zero. In line 6 flow is pushed among the nodes of $C^\oplus$, so by *sources-lemma*$(C^\oplus, C^\oplus \not\to t)$, there are no $C^\oplus$-to-t residual paths after line 6 either. Moreover, by definition of BalancePath, there are no $C^\oplus_+$-to-$C^\oplus_-$ residual paths immediately after line 6 is executed (where $C^\oplus_+$ ($C^\oplus_-$) is the set of nodes of $C^\oplus$ with positive (negative) inflow at that time). Line 7 pushes flow into $C^\oplus_-$, making all the nodes of $C^\oplus_-$ obey conservation. By *sinks-lemma*$(C^\oplus_+ \not\to t, C^\oplus_-)$ there are no $C^\oplus_+$-to-t residual paths upon termination of the procedure. Note that line 7 does not change the set $C^\oplus_+$ as it only pushes as much flow from t to $C^\oplus_-$ as required to make the inflow at the nodes of $C^\oplus_-$ zero. Since $C^\oplus_+$ is the set of nodes of C with positive inflow upon termination, the procedure is correct.

Running time. Note that we can find $C^\oplus$, the set of nodes reachable from $C^+_{\boldsymbol{f}_0}$ via residual paths in linear time. The flow in line 4 can be computed in $O(n \log n)$ time using a single-source single-sink maximum flow algorithm [3]. The *push back* in line 7 can be done in linear time (section 1.4). The running time of the procedure is therefore dominated by the call to BalancePath in line 6, which takes $O(n \log n + |C|^2 \log^2 n)$ time.

5. Extensions: Node capacities and apices. In this section we use the framework of Hochstein and Weihe [23] to extend our algorithm for planar graphs with multiple sources and sinks from the previous sections. First, we give a maximum flow algorithm for directed planar graphs with multiple sources and sinks and with node capacities. The algorithm runs in $O(k^3 n \log^3 n)$ time, where k is the number of nodes with finite capacities. Second, we give a maximum flow algorithm for planar graphs with k apices which also runs in $O(k^3 n \log^3 n)$ time. As far as we know, there is no previous algorithm that exploits planarity to handle node capacities with multiple sources and multiple sinks or apices. The best previously known solution is to use the recent maximum flow algorithm for general graphs of Orlin [41] to get an $O(n^2/\log n)$ time bound for maximum flow in a planar graph with multiple sources, multiple sinks, and node capacities, and $O(kn^2)$ time bound for a graph with k apices.

In the maximum flow problem with *node capacities* we extend the capacity assignment $\boldsymbol{c}(\cdot)$ also to nodes of the input graph, and for each node v we require that $\sum_{a \in E:\mathrm{head}(a)=v} \boldsymbol{f}(a) \le \boldsymbol{c}(v)$. As in the multiple-sources multiple-sinks maximum flow problem, this extension of the maximum flow problem also has a simple reduction which does not preserve the planarity of the graph. Khuller and Naor [29] were the first to study the maximum st-flow in planar graphs with node capacities. Kaplan and Nussbaum [28] showed a reduction from the problem with node capacities to the standard maximum st-flow problem which does preserve the planarity of the graph. However, this reduction works only for a single source and a single sink.

A graph is a *k-apex graph* if there is a subset of k vertices (the *apices*) whose removal leaves a planar graph. We will assume that we know the apices a priori. Chambers and Eppstein [7] suggested that an algorithm for the maximum flow problem in k-apex graphs could be a possible building block towards a maximum flow algorithm for minor-closed families of graphs.

The standard reduction [12] for the maximum flow problem with node capacities replaces each node v with finite capacity with two uncapacitated nodes v_{in} and v_{out}, every arc uv with an arc uv_{in} with the same capacity, and every arc vu with an arc $v_{\mathrm{out}}u$ with the same capacity, and in addition it adds a new arc $v_{\mathrm{in}}v_{\mathrm{out}}$ with capacity $\boldsymbol{c}(v)$. The resulting graph is not planar. However, the set of vertices $\{v_{\mathrm{out}} \mid \boldsymbol{c}(v) < \infty\}$ is an apex set. Therefore, a solution for the maximum flow problem in k-apex graphs implies a solution for the maximum problem in a planar graph with k nodes of finite capacities.

Hochstein and Weihe [23] gave a framework for flow in planar graphs using the Push-Relabel algorithm of Goldberg and Tarjan [17]. We describe this framework here. Let $G = (V, E)$ be a graph with a single source s and a single sink t, where V is divided into two subsets $V^{\times}$ and V_0; $V^{\times}$ are the *stop nodes* and include s and t. Hochstein and Weihe define a complete graph $K^{\times}$ over the stop nodes and compute maximum flow from s to t in $K^{\times}$ using the Push-Relabel algorithm. In short, the Push-Relabel algorithm assigns a label to each node; initially the label of s is n and the labels of the other nodes are 0; the algorithm picks a node with excess inflow, as long as there is such a node, and pushes as much excess inflow as possible from this node to adjacent nodes with lower labels; if there is no way to push the excess from the node the algorithm increases the label of the node. Each time that the Push-Relabel algorithm pushes flow along an arc uv in $K^{\times}$, we implement this step by pushing flow in the original graph G. We do this by pushing flow from u

to v in the subgraph of the residual graph of G induced by $V_0 \cup \{u, v\}$. When the PUSH-RELABEL algorithm on $K^\times$ halts, the flow from s to t in G is maximum. The PUSH-RELABEL algorithm with the FIFO node selection rule for a graph with k nodes requires $O(k^3)$ PUSH steps and $O(k^2)$ RELABEL steps. The RELABEL step consists only of scanning the nodes adjacent to the current node, so all the RELABEL steps can be done in $O(k^3)$ total time. Therefore, the running time of the algorithm of Hochstein and Weihe is $O(k^3 T(n))$, where $T(n)$ is the time to push flow from $u \in V^\times$ to $v \in V^\times$ in the subgraph of G induced by $V_0 \cup \{u, v\}$.

In our case, we let $V^\times$ be the set of apices. We push flow from the source apex u to the sink apex v in the 2-apex graph induced by $V_0 \cup \{u, v\}$ using MSMSMAXFLOW. First, if the dart uv exists in the graph, then saturate this dart and remove it until the end of the PUSH step. We replace the source apex u with a set of source nodes, such that every node that was adjacent to u is adjacent to a unique source of the set. Similarly, we replace the sink apex v with a set of sinks, one for each adjacent node. The resulting graph is planar, and its size remains $O(n)$, so we can find the flow in $O(n \log^3 n)$ time. Note that MSMSMAXFLOW sends a maximum flow, while the amount of flow that the PUSH step pushes is bounded by the excess of the current active node in the PUSH-RELABEL algorithm. However, if the amount of maximum flow is greater than the excess of the active node we can simply push back flow using the procedure of section 1.4.

COROLLARY 5.1. *The maximum flow problem in a directed k-apex graph with n nodes and a given set of k apexes can be solved in $O(k^3 n \log^3 n)$ time.*

COROLLARY 5.2. *The maximum flow problem with multiple sources, multiple sinks, and node capacities in an n-nodes directed planar graph with k nodes with finite capacity can be solved in $O(k^3 n \log^3 n)$ time.*

REFERENCES

[1] R. AHUJA, O. ERGUN, J. ORLIN, AND A. PUNNEN, *A survey of very large scale neighborhood search techniques*, Discrete Appl. Math., 23 (2002), pp. 75–102, https://doi.org/10.1016/S0166-218X(01)00338-9.

[2] G. BORRADAILE, D. EPPSTEIN, A. NAYYERI, AND C. WULFF-NILSEN, *All-pairs minimum cuts in near-linear time for surface-embedded graphs*, in Proceedings of the 32nd International Symposium on Computational Geometry (SoCG), 2016, pp. 22:1–22:16, https://doi.org/10.4230/LIPIcs.SoCG.2016.22.

[3] G. BORRADAILE AND P. N. KLEIN, *An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph*, J. ACM, 56 (2009), pp. 9:1–9:30, https://doi.org/10.1145/1502793.1502798.

[4] Y. BOYKOV AND V. KOLMOGOROV, *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 1124–1137, https://doi.org/10.1109/TPAMI.2003.60.

[5] Y. BOYKOV, O. VEKSLER, AND R. ZABIH, *Fast approximate energy minimization via graph cuts*, IEEE Trans. Pattern Anal. Mach. Intell., 23 (2001), pp. 1222–1239, https://doi.org/10.1109/34.969114.

[6] S. CABELLO, E. W. CHAMBERS, AND J. ERICKSON, *Multiple-source shortest paths in embedded graphs*, SIAM J. Comput., 42 (2013), pp. 1542–1571, https://doi.org/10.1137/120864271.

[7] E. W. CHAMBERS AND D. EPPSTEIN, *Flows in one-crossing-minor-free graphs*, in Proceedings of the 21th Annual International Symposium on Algorithms and Computation (ISAAC), 2010, pp. 241–252, https://doi.org/10.1007/978-3-642-17517-6_23.

[8] S. CORNELSEN AND A. KARRENBAUER, *Accelerated bend minimization*, J. Graph Algorithms Appl., 16 (2012), pp. 635–650, https://doi.org/10.7155/jgaa.00265.

[9] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, 2000, pp. 96–99, https://doi.org/10.1007/978-3-540-77974-2.

[10] J. Erickson, *Maximum flows and parametric shortest paths in planar graphs*, in Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010, pp. 794–804, https://doi.org/10.1137/1.9781611973075.65.

[11] J. Fakcharoenphol and S. Rao, *Planar graphs, negative weight edges, shortest paths, and near linear time*, J. Comput. Syst. Sci., 72 (2006), pp. 868–889, https://doi.org/10.1016/j.jcss.2005.05.007.

[12] C. Ford and D. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[13] P. Gawrychowski, S. Mozes, and O. Weimann, *Improved submatrix maximum queries in Monge matrices*, in Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), 2014, pp. 525–537, https://doi.org/10.1007/978-3-662-43948-7_44.

[14] P. Gawrychowski, S. Mozes, and O. Weimann, *Submatrix maximum queries in Monge matrices are equivalent to predecessor search*, in Proceedings the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), 2015, pp. 580–592, https://doi.org/10.1007/978-3-662-47672-7_47.

[15] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian relation of images*, IEEE Trans. Pattern Anal. Mach. Intell., 6 (1984), pp. 721–742, https://doi.org/10.1109/TPAMI.1984.4767596.

[16] A. Goldberg and S. Rao, *Beyond the flow decomposition barrier*, J. ACM, 45 (1998), pp. 783–797, https://doi.org/10.1145/290179.290181.

[17] A. Goldberg and R. Tarjan, *A new approach to the maximum-flow problem*, J. ACM, 35 (1988), pp. 921–940, https://doi.org/10.1145/48014.61051.

[18] D. Greig, B. Porteous, and A. Seheult, *Exact maximum a posteriori estimation for binary images*, J. Roy. Statist. Soc. B, 51 (1989), pp. 271–279.

[19] R. Hassin, *Maximum flow in (s, t) planar networks*, Inform. Process. Lett., 13 (1981), p. 107, https://doi.org/10.1016/0020-0190(81)90120-4.

[20] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, *Faster shortest-path algorithms for planar graphs*, J. Comput. Syst. Sci., 55 (1997), pp. 3–23, https://doi.org/10.1145/195058.195092.

[21] D. S. Hochbaum, *An efficient algorithm for image segmentation, Markov random fields and related problems*, J. ACM, 48 (2001), pp. 686–701, https://doi.org/10.1145/502090.502093.

[22] D. S. Hochbaum, *The pseudoflow algorithm: A new algorithm for the maximum-flow problem*, Oper. Res., 56 (2008), pp. 992–1009, https://doi.org/10.1287/opre.1080.0524.

[23] J. M. Hochstein and K. Weihe, *Maximum s-t-flow with k crossings in o(k^3n log n) time*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 843–847.

[24] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, *Improved algorithms for min cut and max flow in undirected planar graphs*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC), 2011, pp. 313–322, https://doi.org/10.1145/1993636.1993679.

[25] D. B. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. ACM, 24 (1977), pp. 1–13, https://doi.org/10.1145/321992.321993.

[26] D. B. Johnson and S. M. Venkatesan, *Partition of planar flow networks*, in Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS), 1983, pp. 259–264, https://doi.org/10.1109/SFCS.1983.44.

[27] H. Kaplan, S. Mozes, Y. Nussbaum, and M. Sharir, *Submatrix maximum queries in Monge matrices and partial monge matrices, and their applications*, ACM Trans. Algorithms, 13 (2017), pp. 26:1–26:42, https://doi.org/10.1145/3039873.

[28] H. Kaplan and Y. Nussbaum, *Maximum flow in directed planar graphs with vertex capacities*, Algorithmica, 61 (2011), pp. 174–189, https://doi.org/10.1007/s00453-010-9436-7.

[29] S. Khuller and J. Naor, *Flow in planar graphs with vertex capacities*, Algorithmica, 11 (1994), pp. 200–225, https://doi.org/10.1007/BF01240733.

[30] S. Khuller, J. Naor, and P. Klein, *The lattice structure of flow in planar graphs*, SIAM J. Discrete Math., 6 (1993), pp. 477–490, https://doi.org/10.1137/0406038.

[31] P. N. Klein, *Multiple-source shortest paths in planar graphs*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 146–155.

[32] P. N. Klein, S. Mozes, and O. Weimann, *Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm*, ACM Trans. Algorithms, 6 (2010), pp. 1–18, https://doi.org/10.1145/1721837.1721846.

[33] P. N. Klein and S. Subramanian, *A fully dynamic approximation scheme for shortest paths in planar graphs*, Algorithmica, 22 (1998), pp. 235–249, https://doi.org/10.1007/PL00009223.

[34] J. Kleinberg and E. Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields*, J. ACM, 49 (2002), pp. 616–639, https://doi.org/10.1145/585265.585268.

[35] P. Kohli and P. H. S. Torr, *Dynamic graph cuts for efficient inference in Markov random fields*, IEEE Trans. Pattern Anal. Mach. Intell., 29 (2007), pp. 2079–2088, https://doi.org/10.1109/TPAMI.2007.1128.

[36] J. Lacki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, *Single source–All sinks max flows in planar digraphs*, in Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS), 2012, pp. 599–608, https://doi.org/10.1109/FOCS.2012.66.

[37] G. L. Miller, *Finding small simple cycle separators for* 2*-connected planar graphs*, J. Comput. Syst. Sci., 32 (1986), pp. 265–279, https://doi.org/10.1016/0022-0000(86)90030-9.

[38] G. L. Miller and J. Naor, *Flow in planar graphs with multiple sources and sinks*, SIAM J. Comput., 24 (1995), pp. 1002–1017, https://doi.org/10.1137/S0097539789162997.

[39] S. Mozes and C. Wulff-Nilsen, *Shortest paths in planar graphs with real lengths in* $O(n \log^2 n/ \log\log n)$ *time*, in Proceedings of the 18th European Symposium on Algorithms (ESA), 2010, pp. 206–217, https://doi.org/10.1007/978-3-642-15781-3_18.

[40] Y. Nussbaum, *Network Filow Problems in Planar Graphs*, Ph.D. thesis, Tel-Aviv University, 2014.

[41] J. Orlin, *Max flows in* $O(nm)$ *time, or better*, in Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC), 2013, pp. 765–774, https://doi.org/10.1145/2488608.2488705.

[42] F. R. Schmidt, E. Toeppe, and D. Cremers, *Efficient planar graph cuts with applications in computer vision*, in Proceedings of the 22nd IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 351–356, https://doi.org/10.1109/CVPRW.2009.5206863.

[43] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. Syst. Sci., 26 (1983), pp. 362–391, https://doi.org/10.1016/0022-0000(83)90006-5.

[44] S. Subramanian, *Parallel and Dynamic Shortest-Path Algorithms for Sparse Graphs*, Ph.D. thesis, Brown University, 1995.

[45] X. Wei, A. Joneja, and D. M. Mount, *Optimal uniformly monotone partitioning of polygons with holes*, Comput.-Aided Des., 44 (2012), pp. 1235–1252, https://doi.org/10.1016/j.cad.2012.06.005.