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§
*School of Electrical Engineering and Computer Science, Oregon State University
†Department of Computer Science, Brown University
‡The Blavatnik School of Computer Science, Tel Aviv University
§Department of Mathematics and Computer Science, University of Southern Denmark

***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.**

***Keywords*-maximum flow; planar graph; pseudoflow; planar separator; planar duality;**

I. 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 flow in planar graphs 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) $O(n)$ [13], (b) $O(n \log\log n)$ [16], and (c) $O(n \log n)$ [2], 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 [26]. 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*, which we now know can be solved in planar graphs in $O(n \log^2 n / \log\log n)$ time [27]. Otherwise, Miller and Naor gave an $O(n \log^{3/2} n)$ algorithm for the case where all the sinks and the sources are on the boundary of a single face, and generalized it to an $O(k^2 n^{3/2} \log^2 n)$-time algorithm for the case where the sources and the sinks reside on the boundaries of k different faces.[1]

However, the problem of maximum flow with multiple sources and sinks in planar graphs without any additional restrictions remained open. In general (i.e., non-planar) 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 [26], 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 such as that of Sleator and Tarjan [29] which 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 [9], which leads to a running time of $O(n^{1.5} \log n \log U)$. No planarity-exploiting algorithm was known for the problem.

Theorem I.1. *There exists an algorithm that solves the maximum flow problem with multiple sources and sinks in an n-node directed planar graph in $O(n \log^3 n)$ time.*

Application to computer vision problems: Multiple-source multiple-sink min-cut arises in addressing a family of problems associated with the terms *metric labeling* [24], *Markov Random Fields* [8], and *Potts Model* (see also [4], [14]). 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

GB acknowledges support from NSF Grant CCF-0963921. PNK acknowledges support by NSF Grant CCF-0964037. SM acknowledges support by NSF Grant CCF-0964037 and by a Kanellakis fellowship. YN acknowledges support by the United States - Israel Binational Science Foundation, project number 2006204 and by the Israel Science Foundation grant no. 822-10.

[1]The time bound of the first algorithm can be improved to $O(n \log n)$ using the linear-time shortest-path algorithm of Henzinger et al. [13], and the time bound of the second algorithm can be improved to $O(k^2 n \log^2 n)$ using the $O(n \log n)$-time single-source single-sink maximum flow algorithm of Borradaile and Klein [2].

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. [11]. For the case of more than two labels, there is a powerful and effective heuristic [4] 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 done experimental studies comparing the run-times of different max-flow algorithms on the instances arising in this context (e.g. [3]). 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.[2]

Hochbaum [14] 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. She mentions specifically image segmentation, for which the graph is planar. For this case, by using our algorithm, the optimal solution can be found in nearly linear time.

Application to maximum bipartite matching: It is well known that finding a maximum matching in a bipartite planar graph can be reduced to computing multiple-source, multiple-sink maximum flow. Our result is the first planarity-exploiting algorithm for this problem (and the first near-linear one).

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 [10], [15] and a divide-and-conquer scheme influenced by that of [18] and that of [26]. We adapt a method for using shortest paths to solve max st-flow when s and t are adjacent [12], and a data structure for implementing Dijkstra's algorithm in a dense distance graph derived from a planar graph [7]. Among the other techniques we employ are: using cycle separators [25] recursively while keeping the boundary nodes on a constant number of faces [7], [23], an algorithm for single-source single-sink max flow [2], [6], an algorithm for computing multiple-source shortest paths [5], [22], a method for cancelling cycles of flow in a planar graph [19], and a data structure for range queries in a Monge matrix [20].

[2]Note that the single-source, single-sink max-flow algorithm of [2] was implemented by computer-vision researchers [28] and found to be useful in computer vision and to be faster than the competitors.

A. Preliminaries

We assume the reader is familiar with the basic definitions of planar embedded graphs and their duals (see, e.g., [2]). 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 [2]. The *head* and the *tail* of a dart d are such that d is oriented from $\text{tail}(d)$ to $\text{head}(d)$. We define $\text{rev}(\cdot)$ to be the function that takes each dart to the corresponding dart in the opposite direction. It is notationally convenient to equate the edges, arcs and darts of G with the edges, arcs and darts of the planar dual G^*.

Let $S \subset V$ 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 that satisfies *antisymmetry*: $\boldsymbol{f}(\text{rev}(d)) = -\boldsymbol{f}(d)$ for every dart d. A *capacity assignment* $\boldsymbol{c}(\cdot)$ is a real-valued function on darts. We say a flow assignment $\boldsymbol{f}(\cdot)$ *respects the capacity* of dart d if $\boldsymbol{f}(d) \leq \boldsymbol{c}(d)$, and 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*) at node v is $\text{inflow}_{\boldsymbol{f}}(v) = \sum_{\text{dart } d:\text{head}(d)=v} \boldsymbol{f}(d)$.[3] The *outflow* of v is $\text{outflow}_{\boldsymbol{f}}(v) = -\text{inflow}_{\boldsymbol{f}}(v)$. 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 other than the sources and sinks. A *feasible circulation* is a pseudoflow that obeys conservation at all nodes. The *value* of a feasible flow $\boldsymbol{f}(\cdot)$ is the sum of inflows at the sinks, $\sum_{t \in T} \text{inflow}_{\boldsymbol{f}}(t)$. 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.

A *residual path* in G is a path whose darts all have strictly positive capacities. For two sets of nodes A, B, we write $A \overset{G}{\rightarrow} B$ to state the existence of a residual a-to-b path in G for some nodes $a \in A$ and $b \in B$. Conversely, $A \overset{G}{\not\rightarrow} B$ indicates that no such path exists. We 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 $\boldsymbol{c}_{\boldsymbol{f}}(\cdot)$ such that, for every dart d, $\boldsymbol{c}_{\boldsymbol{f}}(d) = \boldsymbol{c}(d) - \boldsymbol{f}(d)$. It is well-known that a feasible flow $\boldsymbol{f}$ in G is maximum if and only if $S \overset{G_{\boldsymbol{f}}}{\not\rightarrow} T$.

Let $\boldsymbol{f}$ be a pseudoflow in a planar graph G. Let V^+ denote the set of nodes $\{v \in V - (S \cup T) : \text{inflow}_{\boldsymbol{f}}(v) > 0\}$. Similarly, let V^- denote the set of nodes $\{v \in V - (S \cup$

[3]An equivalent definition in terms of arcs is $\text{inflow}_{\boldsymbol{f}}(v) = \sum_{a \in E:\text{head}(a)=v} \boldsymbol{f}(a) - \sum_{a \in E:\text{tail}(a)=v} \boldsymbol{f}(a)$.

$T)$: $\text{inflow}_{\boldsymbol{f}}(v) < 0\}$. Suppose $S \cup V^+ \overset{G_{\boldsymbol{f}}}{\not\to} T \cup V^-$. The pseudoflow $\boldsymbol{f}$ can be converted to a maximum flow in linear time by canceling flow cycles [19] and sending back flow from V^+ and into V^- in topological sort order [18].

B. Overview of the Algorithm

Consider the following recursive approach for finding a max flow: split the input graph G in two using an $O(\sqrt{n})$ simple cycle separator C [25] and recursively solve the max flow problem in each of the two subgraphs. After the recursive calls, each subgraph contains no residual source-to-sink path. We would like to additionally ensure that there is no residual source-to-C/C-to-sink paths in order to ensure that G contains no source-to-sink residual path.

In order to achieve this, we sacrifice conservation; we allow nonzero inflow at nodes of the separator C. To address this, we introduce a new subproblem: pushing flow between the nodes of C so as to ensure that there is no residual path from a positive-inflow node of C to a negative-inflow node of C. Our algorithm for solving this problem (Section III) deals with one node of C at a time. For each node, the algorithm moves flow between that node of C and the others to achieve the following: (i) the node has zero inflow, or (ii) the node has positive inflow but there is no residual path from that node to nodes of C with negative inflow, or (iii) the node has negative inflow but there is no residual path to that node from nodes of C with positive inflow.

Our algorithm for achieving this uses an implicit representation of the flow. This enables it to carry out each iteration in $O(\sqrt{n}\log^2 n)$ time using an efficient data structure [7] for implementing Dijkstra's algorithm to compute shortest paths in the planar dual. Thus the total time for solving this subproblem is $O(n\log^2 n)$, leading to an $O(n\log^3 n)$ running time for the main algorithm. Any omitted proofs can be found in the arXiv version of the paper.

II. The Algorithm

Now we describe the algorithm, MSMSMaxFlow, in more detail. ("MSMS" stands for "multiple-source, multiple-sink".) In order to treat nodes of the cycle separator both as sources and as sinks in recursive calls, we introduce a new node set A of size at most 6. To solve the original problem, one uses $A := \emptyset$.

MSMSMaxFlow($G, \boldsymbol{c}$,S,T,A)
Input: a directed planar graph G with non-negative capacities $\boldsymbol{c}$, a set S of source nodes, a set T of sink nodes, a set A of at most six nodes
Output: a pseudoflow $\boldsymbol{f}$ obeying conservation everywhere except S, T, A and such that $S \overset{G_{\boldsymbol{f}}}{\not\to} T$, $S \overset{G_{\boldsymbol{f}}}{\not\to} A$, $A \overset{G_{\boldsymbol{f}}}{\not\to} T$.

The algorithm finds a size-$O(\sqrt{n})$ simple cycle separator C. Following, e.g., [7], [23], The separator is chosen to ensure each subgraph has at most two-thirds the nodes of G (if $|A| < 6$) or at most two-thirds the nodes of A (if $|A| = 6$). If C contains a source u, introduce an artificial node u' and artificial arc $u'u$ with high capacity, and designate u' a source instead of u. Sinks on C are handled similarly. From now on, we assume for simplicity of presentation that no nodes of C are sources or sinks.

The algorithm contracts all edges of C except one. This merges all the nodes of C into a single supernode v, and turns C into a self-loop. The algorithm then recursively solves the problem on the subgraph embedded on each side of the self-loop, adding v to the set A.

After the recursive calls, the algorithm uncontracts the edges of C. At this stage there are no residual paths between sources and sinks in the entire graph, but there might be positive or negative inflow at nodes of C. The algorithm then calls the procedure FixConservationOnPath, which pushes flow between the nodes of C so that there are no residual paths between nodes of C with positive inflow and nodes of C with negative inflow (the path in the name of the procedure is the cycle C minus one edge). This procedure is discussed in Section III; the interface is:

FixConservationOnPath($G, P, \boldsymbol{c}, \boldsymbol{f}_0$)
Input: a directed planar graph G, a simple path P, a capacity function $\boldsymbol{c}$, and a pseudoflow $\boldsymbol{f}_0$
Output: a pseudoflow $\boldsymbol{f}$ such that $\boldsymbol{f} - \boldsymbol{f}_0$ satisfies conservation everywhere but P, and
$\{v \in P : \text{inflow}(v) > 0\} \overset{G_{\boldsymbol{f}}}{\not\to} \{v \in P : \text{inflow}(v) < 0\}$.
Running Time: $O(n\log n + |P|^2\log^2 n)$

Next, the algorithm iterates over the nodes a_i of A. The algorithm calls a procedure CycleToSinkMaxFlow that pushes as much excess flow as possible from C to a_i. Let C_i^+ 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. CycleToSinkMaxFlow pushes flow among the nodes of C_i^+ and from the nodes of C_i^+ to a_i so that the inflow at every node of C_i^+ remains non-negative and there are no residual paths to a_i from positive-inflow nodes of C. The interface is:

CycleToSinkMaxFlow($G, \boldsymbol{c}, \boldsymbol{f}_0, C, t$)
Input: a directed planar graph G with capacities $\boldsymbol{c}$, a pseudoflow $\boldsymbol{f}_0$, a simple cycle C, a sink t.
Assumes: $\forall v \in C^+$, $\text{inflow}_{\boldsymbol{f}_0}(v) \geq 0$, where $C^+ = \{v \in C : \{x \in C : \text{inflow}_{\boldsymbol{f}_0}(x) > 0\} \overset{G_{\boldsymbol{f}_0}}{\to} v\}$.
Output: a pseudoflow $\boldsymbol{f}$ s.t. *(i)* $\boldsymbol{f} - \boldsymbol{f}_0$ obeys conservation everywhere but $C^+ \cup \{t\}$, *(ii)* $\forall v \in C^+$, $\text{inflow}_{\boldsymbol{f}}(v) \geq 0$, *(iii)* $\{v \in C : \text{inflow}_{\boldsymbol{f}}(v) > 0\} \overset{G_{\boldsymbol{f}}}{\not\to} t$.
Running Time: $O(n\log n + |C|^2\log^2 n)$.

A symmetric procedure, SourceToCycleMaxFlow, is called to push flow from a_i to C to reduce the amount of negative inflow. Finally, the algorithm pushes flow back from any nodes of C with positive inflow to S and pushes flow back from T into any nodes of C with negative inflow.

MSMSMAXFLOW($G, \boldsymbol{c}$,S,T,A)

Input: a directed planar graph G with non-negative capacities $\boldsymbol{c}$, a set S of source nodes, a set T of sink nodes, a set $A = \{a_1, \ldots, a_k\}$ 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 cycle separator C in G
3: ensure that C contains no sources or sinks
4: let $P = C - \{e\}$ where e is one 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 enclosed and not enclosed by e, respectively
7: $\boldsymbol{f}$:= MSMSMAXFLOW($G_1, \boldsymbol{c}_{|G_1}, S \cap G_1, T \cap G_1, (A \cap G_1) \cup \{v\}$)
8: $\boldsymbol{f} := \boldsymbol{f}$ + 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: $\boldsymbol{f}$:= FIXCONSERVATIONONPATH($G, P, \boldsymbol{c}, \boldsymbol{f}$)
11: **for** $i = 1, 2, \ldots, k$
12: $\quad \boldsymbol{f}$:= CYCLETOSINKMAXFLOW($G, \boldsymbol{c}, \boldsymbol{f}, C, a_i$)
13: $\quad \boldsymbol{f}$:= SOURCETOCYCLEMAXFLOW($G, \boldsymbol{c}, \boldsymbol{f}, a_i, C$)
14: push positive excess from C to S and negative excess to C from T
15: **return** $\boldsymbol{f}$

The proof of correctness of MSMSMAXFLOW is given in Section V. It is based on the following lemmata.

Lemma II.1. *(Sources Lemma) Let f be a flow with source set X. Let A, B be two disjoint sets of nodes. Then*

$$A \cup X \overset{G}{\not\to} B \Rightarrow A \overset{G_f}{\not\to} B.$$

Lemma II.2. *(Sinks Lemma) Let f be a flow with sink set X. Let A, B be two disjoint sets of nodes. Then*

$$A \overset{G}{\not\to} B \cup X \Rightarrow A \overset{G_f}{\not\to} B.$$

For node sets W, Y, Z, we will use the notation *sources lemma*(W, Y, Z) to indicate the use of the Sources Lemma to establish that there are no W-to-Y residual paths after a flow with source set Z is pushed. We will use *sinks lemma*(W, Y, Z) in a similar manner.

Running Time Analysis: 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 FIXCONSERVATIONONPATH, CYCLETOSINKMAXFLOW and SOURCETOCYCLEMAXFLOW in G takes $O(|G| \log |G| + |C|^2 \log^2 |C|) = O(|G| \log^2 |G|)$ time. A standard analysis of the recurrence (see, e.g., [7]) shows that the algorithm runs in $O(n \log^3 n)$ time.

III. ELIMINATING RESIDUAL PATHS BETWEEN NODES ON A PATH

In this section we present an efficient implementation of the fixing procedure which, roughly speaking, given a path with nodes having positive, negative, or zero inflow, pushes flow between the nodes of the path so that eventually there are no residual paths from nodes with positive inflow to nodes with negative inflow.

We begin by describing an abstract algorithm for the fixing procedure. It resembles a technique used by Venkatesan and Johnson [18]. Let M be the sum of capacities of all of the darts of G. The algorithm first increases the capacities of darts of the path P and their reverses by M. Let $p_1, p_2, \ldots, p_{k+1}$ be the nodes of P. The algorithm processes the nodes of P one after the other. Processing p_i consists of decreasing the capacities of $d_i = p_i p_{i+1}$ and $rev(d_i)$ by M (i.e., back to their original capacities), and trying to eliminate positive inflow w at p_i by pushing at most w units of flow from p_i to p_{i+1}. After the push, either the flow obeys conservation at p_i or there are no residual paths from p_i to any of the other nodes of P (this is where we use the large capacities on the darts between unprocessed nodes). Negative inflow at p_i is similarly handled by pushing flow from p_{i+1} to p_i. We omit the formal proof of correctness.

ABSTRACTFIXCONSERVATIONONPATH($G, P, \boldsymbol{c}, \boldsymbol{f}_0$)

Input: directed planar graph G, simple path $P = d_1 d_2 \ldots d_k$, capacity function $\boldsymbol{c}$, pseudoflow $\boldsymbol{f}_0$
Output: a pseudoflow $\boldsymbol{f}'$ s.t. *(i)* $\boldsymbol{f}' - \boldsymbol{f}_0$ satisfies conservation at nodes not on P, and *(ii)* with respect to $\boldsymbol{f}'$, there are no residual paths from nodes of P with positive inflow to nodes of P with negative inflow.

1: $\boldsymbol{f}' = \boldsymbol{f}_0$
2: $\boldsymbol{c}[d] = \boldsymbol{c}[d] + M$ for all darts d of $P \cup \text{rev}(P)$
3: **for** $i = 1, 2, \ldots, k$
4: $\quad$ let p_i and p_{i+1} be the tail and head of d_i, respectively
5: $\quad$ // reduce the capacities of d and rev(d) by M and adjust the flow appropriately
6: $\quad$ **for** $d \in \{d_i, \text{rev}(d_i)\}$
7: $\quad\quad \boldsymbol{c}[d] := \boldsymbol{c}[d] - M$
8: $\quad\quad \boldsymbol{f}'[d] := \min\{\boldsymbol{f}'[d], \boldsymbol{c}[d]\}$
9: $\quad\quad \boldsymbol{f}'[\text{rev}(d)] := -\boldsymbol{f}'[d]$
10: $\quad$ *excess* := inflow at p_i
11: $\quad$ **if** *excess* > 0 **then** $d := d_i$ **else** $d := \text{rev}(d_i)$ // find in which direction flow should be pushed
12: $\quad$ add to $\boldsymbol{f}'$ a maximum feasible flow from tail(d) to head(d) with limit *excess*
13: **return** $\boldsymbol{f}'$

A. An Inefficient Implementation

In this section, we give an *inefficient* implementation of line 12 of the abstract algorithm. This will facilitate the

Inefficient Implementation of line 12 of ABSTRACTFIXCONSERVATIONONPATH

1: // push flow on d to make its residual capacity zero as required for Hassin's algorithm
2: *residual capacity* := $\boldsymbol{c}[d] - \boldsymbol{f}[d] - \boldsymbol{\phi}[\text{head}_{G^*}(d)] + \boldsymbol{\phi}[\text{tail}_{G^*}(d)]$
3: *val* := $\min\{$*residual capacity*, $|\text{inflow}(p_i)|\}$ // amount of flow to push on d
4: $\boldsymbol{f}[d] := \boldsymbol{f}[d] +$ *val* ; $\boldsymbol{f}[\text{rev}(d)] := -\boldsymbol{f}[d]$
5: // push excess inflow from tail(d) to head(d) using Hassin's algorithm
6: let $\ell[d'] := \boldsymbol{c}[d'] - \boldsymbol{f}[d'] - \boldsymbol{\phi}[\text{head}_{G^*}(d')] + \boldsymbol{\phi}[\text{tail}_{G^*}(d')]$ for all darts $d' \in G$ // lengths are residual capacities
7: $\ell[\text{rev}(d)] := |\text{inflow}(p_i)|$ // set the limit on the residual capacity of rev(d)
8: $\boldsymbol{\phi}_i(\cdot) :=$ DIJKSTRA$(G^*, \ell, \text{head}_{G^*}(d))$ // face potential are distances in G^* from $\text{head}_{G^*}(d)$ w.r.t. residual capacities
9: *val* := $\boldsymbol{\phi}_i[\text{head}_{G^*}(d)] - \boldsymbol{\phi}_i[\text{tail}_{G^*}(d)]$ // the amount of flow assigned to d by the circulation corresponding to $\boldsymbol{\phi}_i$
10: $\boldsymbol{f}[d] := \boldsymbol{f}[d] -$ *val* ; $\boldsymbol{f}[\text{rev}(d)] := -\boldsymbol{f}[(d)]$ // do not push the circulation on d and rev(d)
11: $\boldsymbol{\phi} = \boldsymbol{\phi} + \boldsymbol{\phi}_i$ // accumulate the current circulation

explanation of the efficient procedure in the next section. We first review the necessary ideas and tools.

1) Hassin's algorithm for maximum st-planar flow: An st-planar graph is a planar graph in which nodes s and t are incident to the same face. Hassin [12] gave an algorithm for computing a maximum flow from s to t in an st-planar graph. We briefly describe our interpretation of this algorithm since we use it in implementing line 12 of the abstract algorithm.

Hassin's algorithm starts by adding to G an artificial infinite capacity arc a from t to s. Let d be the dart that corresponds to a and whose head is t. Let t^* be the head in G^* of the dual of d. Compute in the dual G^* a shortest path tree rooted at t^*, where the length of a dual dart is defined as the capacity of the primal dart. Let $\boldsymbol{\phi}[\cdot]$ denote the shortest path distances from t^* in G^*. Consider the flow

$$\boldsymbol{\rho}[d'] = \boldsymbol{\phi}[\text{head}_{G^*}(d')] - \boldsymbol{\phi}[\text{tail}_{G^*}(d')] \text{ for all darts } d' \quad (1)$$

After removing the artificial arc a from G, $\boldsymbol{\rho}$ is a maximum feasible flow from s to t in G. We say that $\boldsymbol{\phi}$ is a *face potential* vector that induces $\boldsymbol{\rho}$.

In our algorithm we are interested in a *max flow with limit* x from s to t rather than a maximum flow, i.e., a flow whose value is at most a given number x but is otherwise maximal. 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 max flow [21].

Our algorithm, instead of using an artificial arc from t to s, uses an existing arc whose endpoints are s and t as the arc a above. In order for this to work, the capacity of the dart d that corresponds to a and whose head is t must be zero. The algorithm achieves this by first pushing flow on d to saturate it. Also note that our algorithm does not remove a from G. Hence $\boldsymbol{\rho}$ is a feasible circulation, rather than a maximum flow, since flow is being pushed back from t to s along a. To convert it into a maximum flow, the algorithm must remove flow on a. If we define $\boldsymbol{f}$ by

$$\boldsymbol{f}[d'] = \begin{cases} -\boldsymbol{\rho}[d'] & \text{if } d' \text{ corresponds to } a \\ 0 & \text{otherwise} \end{cases}, \quad (2)$$

then $\boldsymbol{\rho} + \boldsymbol{f}$ is a maximum st-flow. We will use the fact that this flow can be represented jointly by the face potential vector $\boldsymbol{\phi}$ and the flow values $\boldsymbol{f}[d']$ for the two darts corresponding to a.

2) The Inefficient Implementation: Recall that $\boldsymbol{f}_0$ is the flow at the beginning of the procedure. Observe that the change to the flow in iteration i of the abstract algorithm (line 12) is a flow between the endpoints of the dart d_i. As discussed in Section III-A1, this flow can be represented as the sum of (i) a circulation $\boldsymbol{\rho}_i$ and (ii) a flow on d_i and rev(d_i). Summing over the first i iterations, the flow $\boldsymbol{f}'$ at that time can be represented as the sum

$$\boldsymbol{f}' = \boldsymbol{\rho} + \boldsymbol{f} \quad (3)$$

where $\boldsymbol{\rho} = \sum_{j=1}^{i} \boldsymbol{\rho}_j$ is a circulation and $\boldsymbol{f}$ is a flow assignment that differs from $\boldsymbol{f}_0$ only on the darts of $\{d_j\}_{j=1}^{i}$ and their reverses.

Now we describe the inefficient implementation of line 12 of ABSTRACTFIXCONSERVATIONONPATH. The total flow $\boldsymbol{f}'$ is maintained by representing $\boldsymbol{f}$ and the circulation $\boldsymbol{\rho}$ as in Eq. (3). $\boldsymbol{f}$ is represented explicitly, but, in preparation for the efficient implementation, the circulations $\boldsymbol{\rho}_j$ are represented implicitly by the face-potentials $\boldsymbol{\phi}_j$. By linearity of Eq. (1), the sum $\boldsymbol{\phi} = \sum_{j=1}^{i} \boldsymbol{\phi}_j$ is the face potential vector that induces the circulation $\boldsymbol{\rho}$.

Recall that d is the dart of C such that flow needs to be sent from tail(d) to head(d) (line 11 of ABSTRACTFIXCONSERVATIONONPATH). In lines 2 – 4, the procedure pushes as much as possible on d itself. Consequently, either d is saturated or conservation at p_i is achieved.

Next, an implementation of Hassin's algorithm pushes a maximum flow with limit $|\text{inflow}(p_i)|$ from tail(d) to head(d). The procedure first sets the length of darts to their residual capacities (line 6) and sets the length of rev(d) to be

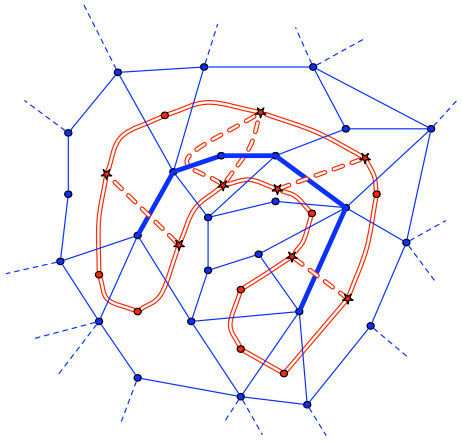

Figure 1. An example illustrating that the nodes of X^* are on the boundary of a single face of H^*. The diagram shows part of the graph G and some edges of its dual G^*. Edges of G are solid blue. Edges of P are bold. Dual edges are double lined red. The dual edges of P are in double lined dashed red. The nodes of X^* are represented by stars.

the flow limit $|\text{inflow}(p_i)|$ (line 7). Since the flow maintained is feasible, all residual capacities are non-negative. The procedure then computes all the $\text{head}_{G^*}(d)$-to-f distances $\boldsymbol{\phi}_i[f]$ in G^* using Dijkstra's algorithm (line 8). Let $\boldsymbol{\rho}_i$ be the circulation corresponding to the face-potential vector $\boldsymbol{\phi}_i$. The procedure sets *val* equal to $\boldsymbol{\rho}_i[d]$ in line 9, then subtracts *val* from $\boldsymbol{f}[d]$ and adds it to $\boldsymbol{f}[\text{rev}(d)]$. Finally, in the last line, the current circulation is added to the accumulated circulation by adding the potential $\boldsymbol{\phi}_i$ to $\boldsymbol{\phi}$.

B. An Efficient Implementation

In this section we give an *efficient* implementation of AbstractFixConservationOnPath. We first review the necessary tools.

1) Fakcharoenphol and Rao's Efficient Dijkstra Implementation: Let X be a set of nodes. Let H be a planar graph in which the nodes of X lie a single face. Let $x_1, x_2, \cdots$ be the clockwise order of the nodes of X on that face. Let P be a set of darts not necessarily in the graph H whose endpoints are nodes in X. Fakcharoenphol and Rao [7] described a data structure that can be used in a procedure that efficiently implements Dijkstra's algorithm in $H \cup P$. The procedure takes as input a table D such that $\text{D}[i,j]$ stores the distance between x_i and x_j in H, an array ℓ that stores the lengths of the darts in P, and a node $v \in X$. It is assumed that the lengths in D and in ℓ are non-negative. The procedure outputs the distances of the nodes of X from v in $H \cup P$ in $O(|X| \log^2 |X| + |P| \log |X|)$-time.

We mention a technical issue whose importance will become apparent for the second tool. The procedure partitions the table D into several subtables $\{\text{D}_\alpha\}_\alpha$ that correspond to distances between pairs of disjoint sets of nodes of X, where each set consists of nodes that are consecutive in X. It is assumed that for each such subtable D_α, a data structure that supports range minimum queries of the form $\min_{j_1 \le j \le j_2}\{\text{D}_\alpha[i,j]\}$ for every i, j_1, j_2 is given [7, footnote on p. 884]. Fakcharoenphol and Rao note that such a data structure can be easily implemented by using a range-search tree for every row i of D_α. The time required to construct all of the range-search trees for D_α is proportional to the size of D_α.

2) Price Functions, Reduced Lengths and FR-Dijkstra: 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 $\ell_\phi(d) = \ell(d) + \phi(\text{tail}(d)) - \phi(\text{head}(d))$. A *feasible* price function is a price function that induces nonnegative reduced lengths on *all* darts of G (see [17]).

Single-source distances form a feasible price function. Suppose that, for some node r of G, for every node v of G, $\phi(v)$ is the r-to-v distance in G with respect to $\ell(\cdot)$. Then for every arc uv, $\phi(v) \le \phi(u) + \ell(uv)$, so $\ell_\phi(uv) \ge 0$. Here we assume, without loss of generality, that all distances are finite (i.e., that all nodes are reachable from r) since we can always add arcs with sufficiently large lengths to make all nodes reachable without affecting the shortest paths in the graph.

We will use the following variant of Fakcharoenphol and Rao's efficient Dijkstra implementation. The procedure $\text{FR}(\text{D}, \ell, \boldsymbol{\phi}^X, v)$ takes as input the table D and the array ℓ as described above, as well as a feasible price function $\boldsymbol{\phi}^X$ on the nodes of X and a node $v \in X$. It outputs the distances of the nodes of X from v in $H \cup P$ w.r.t. the reduced lengths w.r.t. $\boldsymbol{\phi}^X$. We stress that lengths in D and in ℓ may be negative, but the reduced lengths are all non-negative. The computation takes $O(|X| \log^2 |X| + |P| \log |X|)$ time.

This differs from the procedure described in Section III-B1 only in the existence of the price function $\boldsymbol{\phi}^X$. We cannot afford to compute the entire table of reduced lengths since that would dominate the running time of the algorithm in [7]. Instead, whenever their algorithm requires some specific reduced length, we can compute it in constant time from D. This, however, does not suffice. Recall that the algorithm in [7] requires that, for each of the subtables D_α, range-search trees that support range minimum queries of the form $\min_{j_1 \le j \le j_2}\{\text{D}_\alpha[i,j] + \boldsymbol{\phi}^X[x_i] - \boldsymbol{\phi}^X[x_j]\}$ for every i, j_1, j_2 are given. Note that the results of such queries may be different for different price functions. Computing the range-search trees would take time $O(|X|^2)$, which would dominate the running time of the entire procedure. Therefore, we use Monge range-query data structures, due to Kaplan and Sharir [20], which can be constructed from the entire table D in $O(|X| \log^2 |X|)$ time, and answer queries of the desired form in $O(\log |X|)$ time.

3) The Procedure: Finally, we describe the efficient implementation. The procedure keeps track of the inflow at each node of P in an array $\boldsymbol{v}[\cdot]$. As in the inefficient implementation, the procedure will maintain the total flow

Efficient Implementation of line 12 of ABSTRACTFIXCONSERVATIONONPATH

1: // push flow on d to make its residual capacity zero as required for Hassin's algorithm
2: *residual capacity* $:= \boldsymbol{c}[d] - \boldsymbol{f}[d] - \boldsymbol{\phi}^X[\text{head}_{G^*}(d)] + \boldsymbol{\phi}^X[\text{tail}_{G^*}(d)]$
3: *val* $:= \min\{$*residual capacity*$, |\boldsymbol{v}[p_i]|\}$ // amount of flow pushed on d
4: $\boldsymbol{f}[d] = \boldsymbol{f}[d] +$ *val* ; $\boldsymbol{f}[\text{rev}(d)] := -\boldsymbol{f}[d]$
5: $\boldsymbol{v}[\text{tail}(d)] = \boldsymbol{v}[\text{tail}(d)] -$ *val* ; $\boldsymbol{v}[\text{head}(d)] = \boldsymbol{v}[\text{head}(d)] +$ *val* // update the inflow at p_i and p_{i+1}
6: // push excess inflow from tail(d) to head(d) using Hassin's algorithm
7: let $\ell[d'] := \boldsymbol{c}[d'] - \boldsymbol{f}[d']$ for all darts $d' \in P \cup \text{rev}(P)$ // lengths of explicit darts are residual capacities (excluding circulation component)
8: $\ell[\text{rev}(d)] := |\boldsymbol{v}[p_i]| - \boldsymbol{\phi}^X[\text{tail}_{G^*}(\text{rev}(d))] + \boldsymbol{\phi}^X[\text{head}_{G^*}(\text{rev}(d))]$ // limit residual capacity of rev(d) (adjusted by circulation component)
9: $\boldsymbol{\phi}_i^X(\cdot) := \text{FR}(\text{D}^*, \ell, \boldsymbol{\phi}^X, \text{head}_{G^*}(d))$ // distances in G^* from $\text{head}_{G^*}(d)$ w.r.t. the reduced lengths induced by $\boldsymbol{\phi}^X$
10: *val* $:= \boldsymbol{\phi}_i^X[\text{head}_{G^*}(d)] - \boldsymbol{\phi}_i^X[\text{tail}_{G^*}(d)]$ // the amount of flow assigned to d by the circulation corresponding to $\boldsymbol{\phi}_i^X$
11: $\boldsymbol{f}[d] :=$ *val* ; $\boldsymbol{f}[\text{rev}(d)] := -\boldsymbol{f}[(d)]$ // remove the flow assigned by the circulation to d and rev(d)
12: $\boldsymbol{\phi}^X = \boldsymbol{\phi}^X + \boldsymbol{\phi}_i^X$ // accumulate the current circulation
13: $\boldsymbol{v}[\text{tail}_G(d)] = \boldsymbol{v}[\text{tail}_G(d)] -$ *val* ; $\boldsymbol{v}[\text{head}_G(d)] = \boldsymbol{v}[\text{head}_G(d)] +$ *val* // update the inflow at p_i and p_{i+1}

as the sum of a circulation $\boldsymbol{\rho}$ and a flow assignment $\boldsymbol{f}$ that differs from $\boldsymbol{f}_0$ only on the darts of $P \cup \text{rev}(P)$. 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 of $\boldsymbol{\phi}$ that correspond to faces incident to P.

Define each dart d's length by $\ell(d) = \boldsymbol{c}[d] - \boldsymbol{f}[d]$. Let X^* be the set of endpoints in the planar dual G^* of the darts of P (i.e. the primal faces incident to P). Let H^* be the graph obtained from G^* by removing the darts of P. Note that in H^*, all the nodes of X^* that did not disappear (i.e., that have degree greater than zero) are on the boundary of a single face, as illustrated in Figure 1.

The procedure uses a multiple-source shortest paths (MSSP) algorithm [5], [22] to compute a table $\text{D}^*[\cdot,\cdot]$ of X^*-to-X^* distances in H^* with respect to the lengths $\ell(\cdot)$.

The implementation of lines 1– 11 of ABSTRACTFIXCONSERVATIONONPATH using the chosen representation of $\boldsymbol{f}'$ is straightforward. We therefore focus on the implementation of line 12.

Consider iteration i of the algorithm. The main difference between the inefficient implementation and the efficient one is in implementing the Dijkstra step for computing shortest paths in the dual. Instead of computing the entire shortest-path tree, the procedure computes just the distance labels of nodes in X^*. This is done using the FR data structure, whose initialization requires the X^*-to-X^* distances in H^* with respect to the residual capacities. We now explain how these distances can be obtained.

The flow on a dart d in the primal is

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

Therefore the residual capacity of d is

$$(\boldsymbol{c}[d] - \boldsymbol{f}[d]) - \boldsymbol{\phi}[\text{head}_{G^*}(d)] + \boldsymbol{\phi}[\text{tail}_{G^*}(d)] \tag{5}$$

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

Suppose that d belongs to H^*, i.e. d is not one of the darts of $P \cup \text{rev}(P)$. 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

$$\sum_{i=1}^{j} (\ell(d'_i) - \boldsymbol{\phi}[\text{head}_{G^*}(d'_i)] + \boldsymbol{\phi}[\text{tail}_{G^*}(d'_i)]) = \left(\sum_{i=1}^{j} \ell(d'_i)\right) - \boldsymbol{\phi}[\text{end}(Q)] + \boldsymbol{\phi}[\text{start}(Q)]. \tag{6}$$

Therefore, for any nodes $x, y \in X^*$, the x-to-y distance in H^* w.r.t. the residual capacity is given by $\text{D}^*[x,y] - \boldsymbol{\phi}[y] + \boldsymbol{\phi}[x]$. Since the procedure maintains the restriction of $\boldsymbol{\phi}$ to nodes of X^*, this distance can be obtained in constant time.

It follows from the above discussion that, after executing line 12 of ABSTRACTFIXCONSERVATIONONPATH using the efficient implementation for the last time, the flow assignment $\boldsymbol{f}$ maintained by the efficient implementation is the same as the one that would have been computed if the inefficient implementation were used. Moreover, the potential function $\boldsymbol{\phi}^X$ computed by the efficient implementation is the restriction to X^* of the potential function $\boldsymbol{\phi}$ that would have been computed by the inefficient implementation.

As discussed in Section III-A, the flow $\boldsymbol{f}'$ to be returned is $\boldsymbol{\rho} + \boldsymbol{f}$ where $\boldsymbol{\rho}$ is the circulation induced by $\boldsymbol{\phi}$. It suffices, however, to return $\boldsymbol{\chi} + \boldsymbol{f}$ where $\boldsymbol{\chi}$ is any feasible circulation in $G_{\boldsymbol{f}}$: this change does not alter the inflow at any node, and, since it does not change the residual capacity of any cut, does not introduce new residual paths.

To compute $\boldsymbol{\chi}$, the algorithm computes shortest-path distances in the dual of $G_{\boldsymbol{f}}$ from a node $x \in X^*$. This computation consists of two steps. In the first step, the

algorithm again uses FR with price function $\boldsymbol{\phi}$ to compute the distances just to nodes of X^*. In the second step, the algorithm extends this distance labeling to include all nodes of G^*. Any shortest x-to-v path consists of an x-to-x' prefix (where $x' \in X^*$) and a x'-to-v suffix that passes through no nodes of X^*. The distance to v, therefore, is

$$\min_{x' \in X^*} \min\{d(x') + \ell(Q) \ : \\ Q \text{ an } x'\text{-to-}v \text{ path not passing through } X^*\} \quad (7)$$

where $d(x')$ is the x-to-x' distance. Note that every edge of G^* whose endpoints are not both in X^* has nonnegative length. Therefore the distances (7) can be computed by a variant of Dijkstra's algorithm in which each node $x' \in X^*$ is initially inserted into the priority queue with label $d(x')$.

Running Time Analysis: Let n and k be the number of nodes in G and in P, respectively. The initialization time is dominated by the $O(n \log n + k^2 \log n)$ time for MSSP. The execution of each iteration of the main loop is dominated by the call to FR, which takes $O(k \log^2 k)$ time. The number of iterations is $k-1$, leading to a total of $O(k^2 \log^2 k)$ time for execution of the main loop. Computing the circulation $\boldsymbol{\chi}$ requires one more call to FR and one Dijkstra computation in the entire graph, which takes $O(n \log n)$ time. Thus the total running time of the efficient implementation of FIXCONSERVATIONONPATH is $O(n \log n + k^2 \log^2 k)$.

IV. PUSHING EXCESS INFLOW FROM A CYCLE

In this section we describe the procedure CYCLETOSINKMAXFLOW that pushes excess inflow from a cycle to a node not on the cycle.

To compute C^+ in Line 1, consider the residual graph of G w.r.t. $\boldsymbol{f}_0$. Add a node v and non-zero capacity arcs vw for every node w whose inflow w.r.t. $\boldsymbol{f}_0$ is positive (these arcs may not preserve planarity). In $O(|G|)$ time, find the set X of nodes that are reachable from v via darts with non-zero capacity. Then $C^+ = C \cap X$.

Since C^+ consists of all nodes of C reachable via residual paths from the nodes of C with positive inflow, the flow computed by the procedure involves no darts incident to nodes in $C - C^+$. Thus, restricting the computation to the graph obtained from G by deleting the nodes in $C - C^+$ (line 2) does not restrict the computed flow. After deletion, adding artificial arcs between consecutive nodes of C^+ (line 4) will not violate planarity. Contracting the artificial arcs effectively turns C^+ into a single node v_1. Next, the procedure computes a maximum flow $\boldsymbol{f}$ from C^+ to t w.r.t. residual capacities $\boldsymbol{c} - \boldsymbol{f}_0$. This is done by invoking a single-source single-sink maximum flow algorithm [2] with source v_1 and sink t (line 7). Uncontracting the artificial arcs turns $\boldsymbol{f}$ into a maximum C^+-to-t flow in G w.r.t. the residual capacities $\boldsymbol{c} - \boldsymbol{f}_0$. However, some of the nodes of C^+ may have negative inflow w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$. In line 9, the procedure calls FIXCONSERVATIONONPATH to reroute the flow $\boldsymbol{f}$ among the nodes of C^+ so that, w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$, there are no residual paths from nodes of C^+ with positive inflow to nodes of C^+ with negative inflow. This implies that any node of C^+ that still has negative inflow has pushed too much flow. Line 10 modifies $\boldsymbol{f}$ to push back such excess flow so that no node of C^+ has negative inflow w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$. This is done using the technique mentioned in Section I-A. Finally, the procedure returns $\boldsymbol{f}_0 + \boldsymbol{f}$.

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

Input: a directed planar graph G with capacities $\boldsymbol{c}$, a pseudoflow $\boldsymbol{f}_0$, a simple cycle C, a sink t.
Assumes: $\forall v \in C^+$, $\text{inflow}_{\boldsymbol{f}_0}(v) \geq 0$, where $C^+ = \{v \in C : \{x \in C : \text{inflow}_{\boldsymbol{f}_0}(x) > 0\} \overset{G_{\boldsymbol{f}_0}}{\to} v\}$.
Output: a pseudoflow $\boldsymbol{f}$ s.t. *(i)* $\boldsymbol{f} - \boldsymbol{f}_0$ obeys conservation everywhere but $C^+ \cup \{t\}$, *(ii)* $\forall v \in C^+$, $\text{inflow}_{\boldsymbol{f}}(v) \geq 0$, *(iii)* $\{v \in C : \text{inflow}_{\boldsymbol{f}}(v) > 0\} \overset{G_{\boldsymbol{f}}}{\to} t$.

1: let $C^+ := \{v \in C$: there exists a residual path to v from a node $x \in C$ with $\text{inflow}_{\boldsymbol{f}_0}(x) > 0\}$
2: delete the nodes of $C - C^+$
3: let $v_1, v_2, \ldots, v_\ell$ be the nodes of C^+, labeled according to their cyclic order on C
4: add artificial arcs $v_i v_{i+1}$ for $1 \leq i < \ell$
5: let P be the v_1-to-v_ℓ path of artificial arcs
6: contract all the edges of P, collapsing C^+ into the single node v_1
7: $\boldsymbol{f} :=$ ST-MAXFLOW$(G, \boldsymbol{c} - \boldsymbol{f}_0, v_1, t)$
8: undo the contraction of the edges of P
9: $\boldsymbol{f} :=$ FIXCONSERVATIONONPATH$(G, P, \boldsymbol{c}, \boldsymbol{f}_0 + \boldsymbol{f}) - \boldsymbol{f}_0$
10: modify $\boldsymbol{f}$ to push back flow to nodes of C^+ whose inflow w.r.t $\boldsymbol{f}_0 + \boldsymbol{f}$ is negative
11: $\boldsymbol{f} := \boldsymbol{f}_0 + \boldsymbol{f}$
12: **return** $\boldsymbol{f}$

Correctness of CYCLETOSINKMAXFLOW*:* Any flow that is pushed by the procedure originates at C^+ and terminates at $C^+ \cup \{t\}$. Therefore, $\boldsymbol{f} - \boldsymbol{f}_0$ violates conservation only at $C^+ \cup \{t\}$. In line 10, flow of $\boldsymbol{f}$ is pushed back so that no node of C^+ has negative inflow w.r.t. $\boldsymbol{f}_0 + \boldsymbol{f}$. It is possible to do so by only pushing back flow of $\boldsymbol{f}$ (rather than flow of $\boldsymbol{f}_0 + \boldsymbol{f}$) since by assumption no node of C^+ has negative inflow w.r.t. $\boldsymbol{f}_0$.

By maximality of the flow pushed in line 7, just after line 7 is executed there are no C^+-to-t residual paths. This remains true when the edges of P are uncontracted in Line 8. In line 9 flow is pushed among the nodes of C^+, so by *sources lemma*(C^+, t, C^+), there are no C^+-to-t residual paths after line 9 either. Moreover, by definition of FIXCONSERVATIONONPATH, there are no $C_{>0}$-to-$C_{<0}$ residual paths immediately after line 9 is executed, where $C_{>0}$ ($C_{<0}$) is the set of nodes of C^+ with positive (negative) inflow at that time. Line 10 pushes flow into $C_{<0}$,

making all the nodes of $C_{<0}$ obey conservation. By *sinks lemma*$(C_{>0}, t, C_{<0})$ there are no $C_{>0}$-to-t residual paths upon termination. This completes the proof since $C_{>0}$ is the set of nodes of C with positive inflow upon termination.

Running Time Analysis: We next analyze the running time of this procedure on an n-node graph G and a k-node cycle C. The st-maximum flow computation in line 7 takes $O(n \log n)$ time using the algorithm of Borradaile and Klein [2]. The running time of the procedure is therefore dominated by the call to FIXCONSERVATIONONPATH in line 9 which takes $O(n \log n + k^2 \log^2 k)$ time.

V. CORRECTNESS OF MSMSMAXFLOW

We prove that each step of the algorithm eliminates some residual paths without reintroducing residual paths.

Lemma V.1. *At any time in the running of the algorithm after the last execution of line 8 and before the execution of line 14,* $S \nrightarrow T$, $S \nrightarrow A$, $S \nrightarrow C$, $A \nrightarrow T$, $C \nrightarrow T$.

Proof: The properties hold just after line 8 by the properties of the recursive calls and by the fact that any residual path between G_1 and G_2 consists 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.

We next prove that the properties are maintained until just before the execution of line 14. By the above argument, there is a cut separating S from $T \cup A \cup C$ that is saturated just after line 9. The procedure in line 10 only pushes flow between vertices of C, and in lines 11– 13, flow is only pushed between the nodes of A and C. Since these sets are all on the same side of the cut, the cut stays saturated. It follows that $S \nrightarrow \{T, A, C\}$ at any point after line 8 and before line 14. A similar argument applied to a saturated cut separating $A \cup C$ from T shows $A \nrightarrow T$ and $C \nrightarrow T$. ■

Recall that C_i^+ is the set of nodes of C that are reachable via residual paths from some node of C with positive excess at the beginning of iteration i of the loop in line 11, and that C_i^- is the set of nodes of C that have residual paths to some node of C with negative excess at that time.The next lemma follows from the definition of FIXCONSERVATIONONPATH.

Lemma V.2. *Just after line 10 is executed,* $C_1^+ \nrightarrow C_1^-$.

Lemma V.3. *For all* $i < j$, $C_j^+ \subseteq C_i^+$ *and* $C_j^- \subseteq C_i^-$

Proof: It suffices to show that, for all i, $C_{i+1}^+ \subseteq C_i^+$ and $C_{i+1}^- \subseteq C_i^-$. The flow pushed in iteration i of line 12 can be decomposed into a flow whose sources and sinks are all in C_i^+ and a flow whose sources are in C_i^+ and whose sink is a_i. Therefore, the set X of nodes of C with positive inflow immediately after iteration i of line 12 is a subset of C_i^+. By definition of SOURCETOCYCLEMAXFLOW, the set of nodes of C with positive inflow does not change after line 13 is executed. Therefore, C_{i+1}^+ is the set of nodes reachable from X by a residual path after iteration i of line 13.

By definition of C_i^+, immediately before iteration i of line 12 there are no C_i^+-to-$(C - C_i^+)$ residual paths. By *sources lemma*$(C_i^+, C - C_i^+, C_i^+)$, there are no C_i^+-to-$(C - C_i^+$ residual paths immediately after iteration i of line 12 as well. This shows that there are no X-to-$(C - C_i^+)$ residual paths at that time.

The flow pushed in line 13 can be decomposed into a flow whose sources and sinks are all in C_i^- and a flow whose source is a_i and whose sinks are all in C_i^-. By *sinks lemma*$(C_i^+, C - C_i^+, C_i^-)$, there are no C_i^+-to-$(C - C_i^+)$ residual paths immediately after iteration i of line 13. This shows that there are no X-to-$(C - C_i^+)$ residual paths at that time. Hence $C_{i+1}^+ \subseteq C_i^+$, as desired.

The proof of the analogous claim for C_{i+1}^- is similar. ■

Lemma V.4. *Just after line 12 is executed in iteration* i, $C_i^+ \nrightarrow C_i^-$, $C_{i+1}^+ \nrightarrow \{a_j\}_{j \le i}$, $\{a_j\}_{j<i} \nrightarrow C_i^-$.

Proof: The flow pushed in line 12 can be decomposed into a flow whose sources and sinks are all in C_i^+ and a flow whose sources are in C_i^+ and whose sink is a_i.

- $C_i^+ \nrightarrow C_i^-$ by *sources lemma*(C_i^+, C_i^-, C_i^+)
- $C_{i+1}^+ \nrightarrow \{a_j\}_{j<i}$ by *sources lemma*(C_{i+1}^+, a_j, C_i^+)
- $C_{i+1}^+ \nrightarrow \{a_i\}$ by definition of CYCLETOSINKMAXFLOW
- $\{a_j\}_{j<i} \nrightarrow C_i^-$ by *sources lemma*(a_j, C_i^-, C_i^+)

■

The following lemma is proved similarly.

Lemma V.5. *Just after line 13 is executed in iteration* i, $C_i^+ \nrightarrow C_i^-$, $C_{i+1}^+ \nrightarrow \{a_j\}_{j \le i}$, $\{a_j\}_{j \le i} \nrightarrow C_{i+1}^-$.

Let C^+ (C^-) denote the set of nodes in C with positive (negative) inflow just before line 14 is executed.

Corollary V.6. *Just before line 14 is executed,* $C^+ \nrightarrow C^-, C^+ \nrightarrow A, A \nrightarrow C^-$.

Finally, the next lemma proves the algorithm is correct.

Lemma V.7. *The following are true upon termination:*

1) $\boldsymbol{f}$ *is a pseudoflow*
2) $\boldsymbol{f}$ *obeys conservation everywhere except at* S, T, A
3) $S \overset{G_{\boldsymbol{f}}}{\nrightarrow} T, S \overset{G_{\boldsymbol{f}}}{\nrightarrow} A, A \overset{G_{\boldsymbol{f}}}{\nrightarrow} T$.

Proof: Since every addition to $\boldsymbol{f}$ along the algorithm respects capacities of all darts, $\boldsymbol{f}$ is a pseudoflow at all times. By induction, 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, but any such violation is eliminated in line 14. Therefore upon termination $\boldsymbol{f}$ obeys conservation everywhere except S, T and A.

By Lemma V.1 and Corollary V.6, before line 14 is executed, $C^+ \nrightarrow A$ and $C \nrightarrow T$. Therefore the flow pushed back from C^+ in line 14 is pushed back to S. Similarly, the flow pushed back to C^- is pushed back from T. Let f_+ (f_-) be the flow pushed back from C^+ to S (from T to C^-) in line 14. Consider first pushing back f_+.

- $S \nrightarrow T$ by *sources lemma*(S, T, C^+)
- $S \nrightarrow A$ by *sources lemma*(S, A, C^+)
- $A \nrightarrow T$ by *sources lemma*(A, T, C^+)
- $S \nrightarrow C^-$ by *sources lemma*(S, C^-, C^+)
- $A \nrightarrow C^-$ by *sources lemma*(A, C^-, C^+)

Next consider pushing f_-

- $S \nrightarrow T$ by *sinks lemma*(S, T, C^-)
- $S \nrightarrow A$ by *sinks lemma*(S, A, C^-)
- $A \nrightarrow T$ by *sinks lemma*(A, T, C^-)

■

REFERENCES

[1] R. Ahuja, O. Ergun, J. Orlin, and A. Punnen, "A survey of very large scale neighborhood search techniques," *Discrete Appl. Math.*, vol. 23, pp. 75–102, 2002.

[2] G. Borradaile and P. N. Klein, "An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph," *JACM*, vol. 56, no. 2, 2009.

[3] 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.*, vol. 26, no. 9, pp. 1124–1137, 2004.

[4] Y. Boykov, O. Veksler, and R. Zabih, "Efficient approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 12, pp. 1222–1239, 2001.

[5] S. Cabello and E. W. Chambers, "Multiple source shortest paths in a genus g graph," in *18th SODA*, 2007, pp. 89–97.

[6] J. Erickson, "Maximum flows and parametric shortest paths in planar graphs," in *21st SODA*, 2010, pp. 794–804.

[7] J. Fakcharoenphol and S. Rao, "Planar graphs, negative weight edges, shortest paths, and near linear time," *J. Comput. Syst. Sci.*, vol. 72, no. 5, pp. 868–889, 2006.

[8] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian relation of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–742, 1984.

[9] A. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *JACM*, vol. 45, no. 5, pp. 783–797, 1998.

[10] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by successive approximation," *Math. Oper. Res.*, vol. 15, no. 3, pp. 430–466, 1990.

[11] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *J. Roy. Stat. Soc. B*, vol. 51(2), pp. 271–279, 1989.

[12] R. Hassin, "Maximum flow in (s, t) planar networks," *Inform. Process. Lett.*, vol. 13, no. 3, p. 107, 1981.

[13] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 3–23, 1997.

[14] D. S. Hochbaum, "An efficient algorithm for image segmentation, Markov random fields and related problems," *JACM*, vol. 48, no. 4, pp. 686–701, 2001.

[15] ——, "The pseudoflow algorithm: A new algorithm for the maximum-flow problem," *Operations Research*, vol. 56, no. 4, pp. 992–1009, 2008.

[16] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, "Improved algorithms for min cut and max flow in undirected planar graphs," in *43rd STOC*, 2011, pp. 313–322.

[17] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *JACM*, vol. 24, no. 1, pp. 1–13, 1977.

[18] D. B. Johnson and S. Venkatesan, "Using divide and conquer to find flows in directed planar networks in $O(n^{3/2} \log n)$ time," in *Proc. of the 20th Annual Allerton Conf. on Communication, Control, and Computing*, 1982, pp. 898–905.

[19] H. Kaplan and Y. Nussbaum, "Maximum flow in directed planar graphs with vertex capacities," in *17th ESA*, 2009, pp. 397–407.

[20] H. Kaplan and M. Sharir, "Finding the maximal empty rectangle containing a query point," 2011, arXiv:1106.3628v1 [cs.CG].

[21] S. Khuller, J. Naor, and P. Klein, "The lattice structure of flow in planar graphs," *SIAM J. Discret. Math.*, vol. 6, no. 3, pp. 477–490, 1993.

[22] P. N. Klein, "Multiple-source shortest paths in planar graphs," in *16th SODA*, 2005, pp. 146–155.

[23] P. N. Klein and S. Subramanian, "A fully dynamic approximation scheme for shortest paths in planar graphs," *Algorithmica*, vol. 22, no. 3, pp. 235–249, 1998.

[24] J. Kleinberg and E. Tardos, "Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields," in *40th FOCS*, 1999.

[25] G. L. Miller, "Finding small simple cycle separators for 2-connected planar graphs," *J. Comput. Syst. Sci.*, vol. 32, no. 3, pp. 265–279, 1986.

[26] G. L. Miller and J. Naor, "Flow in planar graphs with multiple sources and sinks," *SIAM J. Comput.*, vol. 24, no. 5, pp. 1002–1017, 1995, preliminary version in FOCS 1989.

[27] 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 *18th ESA*, 2010, pp. 206–217.

[28] F. R. Schmidt, E. Toeppe, and D. Cremers, "Efficient planar graph cuts with applications in computer vision," in *CVPR*, 2009, pp. 351–356.

[29] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, 1983.