An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph*

Glencora Borradaile† Philip Klein‡

Abstract

We give the first correct $O(n \log n)$ algorithm for finding a maximum st-flow in a directed planar graph. After a preprocessing step that consists in finding single-source shortest-path distances in the dual, the algorithm consists of repeatedly saturating the leftmost residual s-to-t path.

1 Introduction

Informally, the *maximum st-flow problem* is as follows: given a graph with positive arc-capacities, and given a source vertex s and a sink vertex t, the goal is to find a way to route the maximum amount of a single commodity along s-to-t paths in such a way that the amount of commodity passing through an arc is at most the capacity of the arc. In the *minimum st-cut problem*, the goal is to find a minimum-capacity set of arcs such that each s-to-t path includes at least one arc in the set. Formal definitions will be given in Section 4.5.

The history of maximum-flow and minimum-cut problems [32] is tied closely to planar graphs. During the height of the cold war, the United States spent considerable effort analyzing the Soviet rail network: "The success of interdiction depends largely on [the] interdiction-program efforts on the enemy's capability to move men and supplies." [16] Modeling the Soviet rail network as a planar graph (by taking the dual of the planar graph composed of boundaries of administrative districts with edges representing transportation capacity between these districts), Harris and Ross, as members of the RAND corporation, studied the problem of determining the best way to interdict the Soviet rail network. That is, they found the minimum number of rail lines that must be cut in order to stop movement of supplies and men between tactically important locations: they found a minimum cut in the graph. Ford and Fulkerson picked up on this line of research, leading to their landmark paper proving the max-flow, min-cut theorem and formulating the augmenting-path algorithm [11].

*A preliminary version of this paper was published as [4].

†Department of Combinatorics and Optimization, University of Waterloo. Work done while at the Department of Computer Science, Brown University. Partially supported by the Rosh foundation, a PGS-D fellowship from NSERC, and NSF Grant CCF-0635089.

‡Department of Computer Science, Brown University. Partially supported by NSF Grant CCF-0635089.

Theorem 1.1 (Max-Flow Min-Cut)**.** *The value of the maximum st-flow is equal to the capacity of the minimum st-cut.*

The Max-Flow Min-Cut Theorem was discovered independently by Ford and Fulkerson [11], Kotzig [28], and Elias et al. [8].

2 Maximum flow algorithms

In the same paper that proved the Max-Flow Min-Cut Theorem, Ford and Fulkerson suggested an algorithm (actually, a paradigm) for finding a maximum flow called the *augmenting path algorithm*. The algorithm is iterative: find a path P from the source to the sink and *push* flow on this path. The residual capacity of an arc is the capacity less the flow on the arc. That is, the value of the flow for each dart in P is increased by an amount Δ that does not exceed the residual capacity of any arc on P. A formal description will be given in Section 4.5.

Dinitz [5] and Edmonds and Karp [7] showed that if the shortest (with respect to number of arcs) augmenting path is chosen then there are at most nm iterations. Dinitz gave an $O(n^2m)$ analysis for this using the notion of a blocking flow. In [14], Goldberg and Rao gave a clever implementation resulting in an $O(\min(n^{2/3}, m^{1/2})m \log \frac{n^2}{m} \log U)$-time algorithm (where U is the largest integral capacity) by using a different, adaptive notion of distance that is related to the residual capacities. This is the fastest known algorithm for maximum flow in a general graph and results in an $O(n^{3/2} \log n \log U)$-time algorithm for planar graphs. For a more detailed survey see [13].

We briefly mention another type of maximum flow algorithm: the push-relabel algorithm, alternatively known as the preflow-push algorithm [15]. Rather than pushing flow along paths, flow is pushed on individual arcs. This algorithm does not maintain a feasible flow during its execution; it augments arcs in order to bring the flow closer to feasibility.

3 History of planar maximum flow

In [11], Ford and Fulkerson gave a particular augmenting-path algorithm for the case of finding the maximum st-flow in a planar graph in which the source and the sink are on the boundary of a common face, the infinite face. Such a graph is termed *st-planar*. With the graph viewed with the source embedded on the left and the sink on the right, the algorithm iteratively augments (in fact, *saturates*) the *uppermost* residual path. This algorithm has the property that the flow on an arc is never decreased. Since each augmentation makes at least one arc non-residual, the algorithm requires at most m augmentations, where m is the number of arcs. In 1979, Itai and Shiloach [22] showed that each iteration of the uppermost path algorithm could be implemented in $O(\log n)$ time, where n is the number of vertices, using a priority queue of the residual

darts. Consequently, the algorithm can be carried out in $O(n \log n)$ time (using the fact that a simple planar graph with n vertices has at most $3n$ arcs).

In 1981, Hassin demonstrated that a maximum st-flow in an st-planar graph G could be derived from shortest-path distances in the planar dual G^* (see Section 4.3) of G where capacities in G are interpreted as lengths in G^*. With this insight, it can be seen that the uppermost-path algorithm can be interpreted in the planar dual as Dijkstra's algorithm. The fact that the uppermost path algorithm can be implemented to run in $O(n \log n)$ time corresponds to the observation, due to Johnson [23], that Dijkstra's algorithm could be implemented to run in $O(n \log n)$ time by using a priority queue. Frederickson showed later that shortest-path distances in a planar graph with nonnegative lengths could be computed in $O(n\sqrt{\log n})$ time [12], and Henzinger et al. showed subsequently that the same problem could be solved in $O(n)$ time [21]. Combining this with Hassin's result yields an $O(n)$-time algorithm for maximum st-flow in st-planar graphs.

There remained, however, the more general and more natural problem of st-flow in a planar graph in which s and t need not be on the boundary of a common face. In 1983, Reif [30] showed that the minimum st-cut (and so, via the Max-Flow Min-Cut Theorem, also the value of the max st-flow) could be found in $O(n \log^2 n)$ time for the special case of *undirected* planar graphs. This algorithm uses the observation that the edges crossing a min st-cut form a minimum length cycle C that separates s from t in the planar dual graph (where s and t are faces). The algorithm finds a shortest path P in the planar dual from a vertex incident to s to a vertex incident to t. Reif proves that C only crosses P once. A divide-and-conquer algorithm is given in which a minimum separating cycle is found that contains the middle edge e of P: this cycle corresponds to a min cut in the primal separating the endpoints of e. This results in an $O(n \log^2 n)$-time algorithm, using the aforementioned $O(n \log n)$ st-planar flow algorithm of Itai and Shiloach. In 1985, Hassin and Johnson [18] drew on Reif's technique to show that the flow assignment could also be found within the same time bound, again for *undirected* planar graphs. The shortest-path algorithms of Henzinger et al. [21] or Klein [27] can be used to re-implement these algorithms in $O(n \log n)$ time.

Still the more general problem of st-flow in a planar directed graph remained open. This problem is more general since the problem of maximum st-flow in an undirected graph can be converted to a directed problem by introducing two oppositely oriented arcs of equal capacity for each edge. In 1982, Johnson and Venkatesan gave a divide-and-conquer algorithm that finds a flow of input value v in a directed planar graph in $O(n^{1.5} \log n)$ time [24]. The algorithm divides the graph using an $O(\sqrt{n})$-balanced separator, and recursively finds a flow in each side of value v. The flow on the $O(\sqrt{n})$-boundary edges of each subproblem might not be feasible; each boundary edge is made feasible via an st-planar flow computation.

In 1989, Miller and Naor [29] showed that finding a maximum st-flow can be reduced to a sequence of shortest-path computations in a graph with positive and negative lengths. In 2001, Fakcharoenphol and Rao [10] presented

an $O(n \log^3 n)$ algorithm for the latter problem, implying an $O(n \log^3 n \log C)$ bound on maximum flow where C is the sum of the integral capacities.

Year	**Restriction**	**Time**	**Reference**
1956	*st*-planar	$O(n^2)$	Ford and Fulkerson [11]
1979	*st*-planar	$O(n \log n)$	Itai and Shiloach [22]
1982	fixed value	$O(n\sqrt{n} \log n)$	Johnson and Venkatesan [24]
1983	value undirected	$O(n \log^2 n)$	Reif [30]
1985	undirected	$O(n \log^2 n)$	Hassin and Johnson [18]
1987	*st*-planar	$O(n\sqrt{\log n})$	Hassin [17] using Frederickson [12]
1997	*st*-planar	$O(n)$	Hassin [17] using Henzinger et al. [21]
1997	undirected	$O(n \log n)$	Hassin and Johnson [18] using Henzinger et al. [21]
2001		$O(n \log^3 n \log C)$	Miller and Naor [29] using Fakcharoenphol & Rao [10]

Table 1: Planar Maximum-Flow and Minimum-Cut Algorithms

3.1 Toward an $O(n \log n)$ algorithm

In 1994, Weihe [36] published an $O(n \log n)$ algorithm for planar directed maximum *st*-flow. The algorithm, though perhaps influenced by Ford and Fulkerson's uppermost-path algorithm, is quite different. From the example included in his paper, it is clear that an augmenting path is not necessarily an uppermost path (as generalized to non-*st*-planar graphs). The algorithm and proof of correctness are quite complicated.

In a preprocessing step of the algorithm, the input graph is transformed into one satisfying the following three requirements.

1. Each vertex but the source and sink has degree exactly three;
2. there are no clockwise cycles; and
3. each arc uv belongs to a simple s-to-v path and a simple u-to-t path.

Satisfying Requirement 1 involves: splicing together every two successive arcs sharing an endpoint of total degree two; and replacing each vertex of high degree by a cycle, increasing the number of vertices to $2m$, which is at most $6n$. Requirement 2 can be satisfied by using a reduction of Khuller, Naor, and Klein [26] to computing shortest-path distances in the dual (and so can be computed in $O(n)$ time using the algorithm of Henzinger et al. [21]). Details of this step will be given in Section 5.1.

Requirement 3 is problematic. Weihe states "To satisfy this assumption, simply remove all arcs that violate it. None of these arcs will help us solve

our problem." However, as pointed out by Biedl, Brejová, and Vinař [3], there is no known $O(n \log n)$-time algorithm to delete all such arcs. They give the best known algorithm to date, which runs in $O(n^2)$ time. To our knowledge, the dependence of Weihe's proof of correctness on Requirement 3 has not been resolved. Although Weihe has claimed that his algorithm can be corrected, this has not been verified.

4 Preliminaries

4.1 Graphs

Let G be a directed graph with arc-set A. For each arc $a \in A$, 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.

We define $rev(\cdot)$ to be the function that takes each dart to the corresponding dart in the opposite direction. Formally, the dart set is $A \times \{\pm 1\}$, and $rev(\langle a, i\rangle) = \langle a, -i\rangle$. The head and tail of a dart d in a graph G (written $head_G(d)$ and $tail_G(d)$) are such that the dart is oriented from tail to head. We may omit the subscript when doing so introduces no ambiguity. We may use uv to indicate a dart d such that $u = tail(d)$ and $v = head(d)$ when there is no ambiguity due to parallel darts.

Walks, Paths and Cycles

A nonempty x-to-y *walk* is a nonempty sequence of darts $d_1 \dots d_k$ such that $head_G(d_1) = x$, $tail_G(d_k) = y$ and, for $i = 2, \dots, k$, $head_G(d_{i-1}) = tail_G(d_i)$. The walk is said to *contain* a vertex if the vertex is the head or tail of one of the darts making up the walk. The *start vertex* of the walk is defined to be the tail of d_1, and the *end vertex* is defined to be the head of d_k. We denote the start and end vertices of a walk W by $start(W)$ and $end(W)$, respectively. An empty walk W is specified by a single vertex v such that $start(W) = end(W) = v$.

A walk in which no dart appears more than once is a *path*. If in addition $head_G(d_k) = tail_G(d_1)$ then the path is a cycle. A path/cycle of darts is *simple* if no vertex occurs twice as the head of a dart in the path/cycle. A path/cycle is said to be *directed* if each of its darts has the same orientation as the corresponding arc. We use the term *undirected* when we wish to emphasize that a path/cycle need not be directed. A graph or subgraph is *connected* if for every pair u, v of vertices it contains an undirected u-to-v path.

Subpaths and path concatenation

For a walk W containing vertices u and v, $W[u, v]$ denotes the u-to-v subwalk of W. (If u or v occurs more than once in W, we will use this notation only when it is clear which occurrence is intended.) If W is a cycle, $W[u, v]$ denotes a subpath of the cycle.

$W[\cdot, v]$ is shorthand for $W[start(W), v]$, and $W[u, \cdot]$ is shorthand for $W[u, end(W)]$. We use $W[u, v)$ to denote the walk obtained from $W[u, v]$ by deleting the last dart; $W(u, v]$ and $W(u, v)$ are defined similarly. The reverse of $W = d_1 \dots d_k$, denoted by $rev(W)$, is the walk $rev(d_k)\ rev(d_{k-1})\ \dots\ rev(d_1)$.

If $P = d_1 \dots d_k$ and $Q = e_1 \dots e_\ell$ are walks such that $end(P) = start(Q)$, we use $P \circ Q$ to denote the walk $d_1 \dots d_k e_1 \dots e_\ell$.

Subgraphs

A subgraph H of a graph G is identified with a subset of arcs.

A *spanning tree* T of G is a connected subgraph of G that contains all vertices of G and contains no undirected simple cycle. For vertices u and v, $T[u, v]$ denotes the unique (undirected) u-to-v path through T.

If a vertex of T is designated as a root, we use $T[u]$ to denote the u-to-root path in T. Let xy be an edge of T where $T[x]$ includes y. Then the corresponding dart, oriented from x to y, is called the *parent dart* of x in T.

For a graph G and a set S of vertices of G, $\delta^+_G(S)$ is the set of darts whose tails are in S and whose heads are not. (We omit the subscript when the choice of graph is clear from the context.) Such a set of darts is called a *cut*. A cut is a *simple cut* if both S and $V \setminus S$ are connected. A simple cut is also known as a *bond*.

4.2 Vector spaces

The *arc space* of a graph $G = \langle V, A\rangle$ is the vector space $\mathbf{R}^A$: a vector $\boldsymbol{\delta}$ in arc space assigns a real number $\boldsymbol{\delta}[a]$ to each arc $a \in A$. It is notationally convenient to interpret a vector $\boldsymbol{\delta}$ in arc space as assigning real numbers to all darts. For a dart $\langle a, i\rangle$ ($i = \pm 1$), we define

$$\boldsymbol{\delta}[\langle a, i\rangle] = i \cdot \boldsymbol{\delta}[a].$$

That is, if d is a dart in the same direction as the corresponding arc a then $\boldsymbol{\delta}[d] = \boldsymbol{\delta}[a]$, and if d points in the opposite direction then $\boldsymbol{\delta}[d] = -\boldsymbol{\delta}[a]$.

For each arc a, we define $\boldsymbol{\delta}(a)$ to be the vector in arc space that assigns 1 to a and zero to all other arcs:

$$\forall a' \in A,\ \boldsymbol{\delta}(a)[a'] = \begin{cases} 1 & \text{if } a' = a, \\ 0 & \text{otherwise.} \end{cases}$$

For a multi-set S of darts, we define $\boldsymbol{\delta}(S) = \sum_{d \in S} \boldsymbol{\delta}(d)$.

The *cycle space* of G is the subspace of the arc space spanned by

$$\mathcal{C} = \{\boldsymbol{\delta}(C)\ :\ C \text{ a cycle of darts in } G\}.$$

We will refer to vectors in cycle space alternatively as *circulations*.

4.3 Planar graphs

According to the geometric definition, a planar graph is a graph for which there exists a planar embedding. A planar embedding of a graph is a drawing of the graph on the plane (or the surface of a sphere) so that vertices are mapped to distinct points and edges are mapped to non-crossing curves. A *face* is a connected component in the set of points that are not in the image of any arc or vertex. For a connected graph embedded on the plane, there is one *infinite face*. For a connected graph embedded on the sphere, an arbitrary face can be designated as the infinite face. We denote the infinite face by f_∞.

One can alternatively define embeddings combinatorially, without reference to topology [19, 6, 38]. A combinatorial embedding is sometimes called a *rotation system*. A combinatorial embedding is given by a permutation π such that for each dart d, $\pi(d)$ is the dart e such that $x = tail(d) = tail(e)$ and e is the dart immediately after d in the counterclockwise ordering of the darts around x. While such a formulation frequently makes the implementation of algorithms simpler, we will only use the permutation π explicitly in a few places throughout this work. However, we note that for all the algorithms contained herein, a combinatorial embedding is sufficient for implementation.

Duals of planar graphs

Corresponding to every connected planar embedded graph G there is another connected planar embedded graph denoted G^*. The faces of G are the vertices of G^* and vice versa. The arcs (and hence darts) of G correspond one-to-one with those of G^*. If d is a dart of G, the tail of the corresponding dart of G^* is the face to the left of d, and the head is the face to the right of d. Thus intuitively the geometric orientation in G^* of the dart corresponding to d is obtained by rotating the embedding of d clockwise roughly 90 degrees. It is notationally convenient to equate the darts of G with the darts of G^*. We call G the primal graph and G^* the dual. An example is given in Figure 1. The rotation system for the combinatorial embedding of G^* is denoted π^* and is equal to $\pi \circ rev$.

The *boundary of a face* f of a planar embedded graph is the cycle consisting of darts whose tail in the planar dual is f, ordered according to the cycle of π^* corresponding to f. Figure 1 gives an example. We denote the boundary of f by ∂f. The *boundary* of the planar embedded graph G is denoted by ∂G and is defined to be the boundary of its infinite face. According to our convention, for a face f other than the infinite face, ∂f is oriented counterclockwise, and ∂f_∞ is oriented clockwise.

We will liberally use the following two classical results on planar graphs. These theorems are illustrated in Figures 2 and 3, respectively.

Theorem 4.1 (Interdigitating Spanning Trees [9, 34])**.** *For a spanning tree T of G, the set of arcs* not *in T form a spanning tree of the dual G^*.*

We denote the set of arcs not in T by T^*.

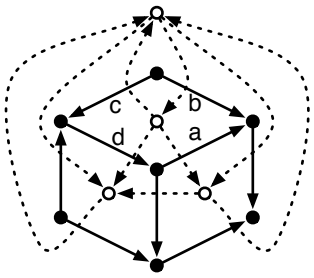


Figure 1: A planar graph and its dual: the primal is given by solid vertices and solid arcs and the dual is given by open vertices and dotted arcs. The boundary of one of the faces is the cycle $a\ rev(b)\ c\ d$.

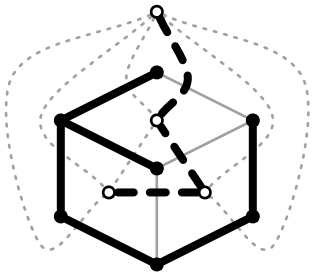


Figure 2: The primal is given by solid arcs and the dual by dotted arcs. The dark bold edges form a spanning tree T of the primal. The edges not in T form a spanning tree T^* of the dual.

Theorem 4.2 (Cycle-Cut Duality [37])**.** *In a connected planar graph, a set of darts forms a simple directed cycle in the primal iff it forms a simple directed cut in the dual.*

Circulations

Recall that the *cycle space* of G is the subspace of the arc space spanned by $\mathcal{C} = \{\boldsymbol{\delta}(C) \ : \ C \text{ a cycle of darts in } G\}$. Recall that ∂f consists of the darts making up the boundary of face f. In a connected planar graph, the set of vectors

$$\{\boldsymbol{\delta}(\partial f) \ : \ f \text{ a face of } G, f \neq f_\infty\}$$

is a basis for the cycle space of G. Therefore, any vector $\boldsymbol{\eta} \in \mathcal{C}$ can be represented as a linear combination of these basis vectors. We use $\boldsymbol{\phi}$ to denote the vector of coefficients for this linear combination, so

$$\boldsymbol{\eta} = \sum_{f \neq f_\infty} \boldsymbol{\phi}[f]\boldsymbol{\delta}(\partial f)$$

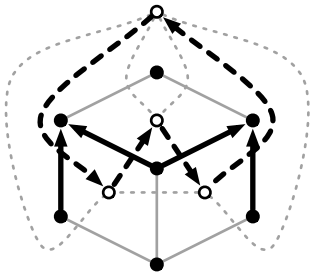


Figure 3: The primal is given by solid edges and the dual by dotted edges. The dark bold (directed) darts form a simple directed cycle in the dual and a directed bond, $\delta^+(S)$, in the primal, where S is the set of the lower 4 vertices.

We call $\boldsymbol{\phi}$ a *potential assignment*, and we refer to $\boldsymbol{\phi}[f]$ as the *potential* of face f. This use of potentials was introduced by Hassin [17] for st-planar graphs, and by Miller and Naor [29] for general planar graphs. We adopt the convention that $\boldsymbol{\phi}[f_\infty] = 0$.

Encloses

For a simple cycle C with a geometric embedding, we can say that C *strictly encloses* a dart, vertex or face if the dart, vertex or face is embedded inside C with respect to f_∞. We formalize this definition for any cycle C (not necessarily simple).

Let C be a cycle and let $\boldsymbol{\phi}$ be the potential corresponding to the circulation $\boldsymbol{\delta}(C)$. Cycle C *encloses* a face f if $\boldsymbol{\phi}[f] \neq 0$. Cycle C strictly encloses a dart d if C encloses the faces to the left and right of d ($tail_{G^*}(d)$ and $head_{G^*}(d)$, respectively). Cycle C encloses a dart d if C strictly encloses d, $d \in C$ or $rev(d) \in C$. Cycle C encloses or strictly encloses a vertex v if C encloses or strictly encloses, respectively) all the darts incident to v.

Clockwise and Counterclockwise

A circulation is defined as *counterclockwise* (abbreviated *c.c.w.*) if the potential of every face is nonnegative [26]. A circulation is defined as *clockwise* (abbreviated *c.w.*) if the potential of every face is non-positive. A directed cycle C of darts is clockwise if $\boldsymbol{\delta}(C)$ is clockwise. A cycle or circulation may be neither counterclockwise nor clockwise, but a simple cycle is either clockwise or counterclockwise. These definitions have geometric interpretations, as illustrated by Figure 4.

For any face f, the boundary of f is a clockwise cycle if $f \neq f_\infty$ and a counterclockwise cycle if $f = f_\infty$.

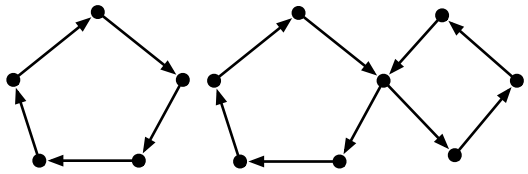


Figure 4: (a) A clockwise cycle. (b) A cycle that is neither clockwise nor counterclockwise.

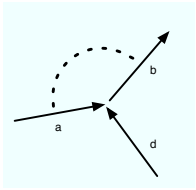


Figure 5: The dart d enters the path $a \circ b$ from the right because d does not appear among the sequence of darts in counterclockwise order between b and $rev(a)$.

Entering and Crossing

The notions of *entering* and *crossing* are illustrated in Figures 5 and 6.

Suppose a, b, and d are darts such that $head(a) = tail(b) = head(d)$: we say d *enters* $a \circ b$ at $head(a)$. If in addition $rev(d)$ is not among $b, \pi(b), \pi(\pi(b)), \ldots, \pi^k(b)=rev(a)$ then d is said to enter $a \circ b$ *from the right* at $head(a)$. (See Figure 5.) A path that contains d is said to enter a walk that contains $a \circ b$ from the right. *Enters from the left*, *leaves from the right*, etc., are defined similarly.

Suppose paths P and Q are such that R is a maximal common subpath of P and Q. We say that P *crosses* Q if

- P enters Q from the right at $start(R)$ and P leaves Q from the left at $end(R)$, or
- P enters Q from the left at $start(R)$ and P leaves Q from the right at $end(R)$.

If P and Q are paths that do not cross, then they are *non-crossing*. A path/cycle is *non-self-crossing* if for every pair P and Q of subpaths of the path, P does not cross Q. Note that, for any face f, the boundary of f is a non-self-crossing cycle.

We will use the following lemma in Section 6 where we will use non-self-crossing cycles (simple cycles are not sufficiently general). This lemma allows us to build non-self-crossing cycles from other non-self-crossing cycles.

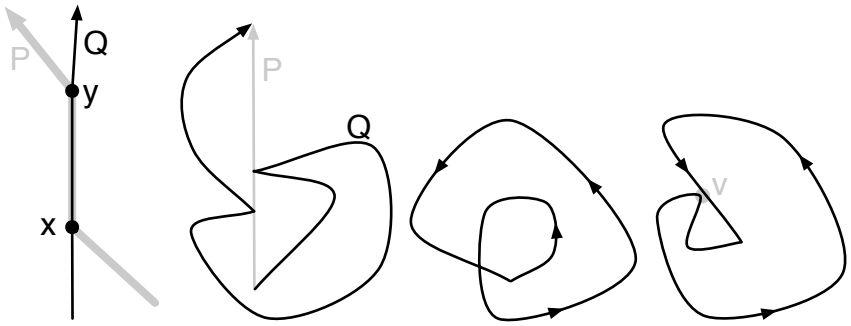


Figure 6: (a) P crosses Q: P enters Q on the right at x and P leaves Q on the left at y. (b) P and Q are non-crossing. (c) This is a self-crossing cycle. (d) This is a non-self-crossing but non-simple cycle, since vertex v occurs twice.

Lemma 4.3 (Composition Lemma)**.** *Let C be a non-self-crossing cycle and let A be a non-self-crossing path with endpoints on C such that no part of A is enclosed by C. Then*

$$A \circ C[end(A), start(A)]$$

is a non-self-crossing cycle.

Proof. Since no part of A is enclosed by C, A does not cross C. It follows that $A \circ C[end(A), start(A)]$ is non-self-crossing. □

4.4 Clockwise and leftmost

An x-to-y walk A is *left of* an x-to-y walk B if $\boldsymbol{\delta}(A) - \boldsymbol{\delta}(B)$ is a clockwise circulation. (This definition was given by [27] for paths, but generalizes naturally to walks.) Likewise A is *right of* B if $\boldsymbol{\delta}(A) - \boldsymbol{\delta}(B)$ is a counterclockwise circulation. *Left of* and *right of* are transitive, reflexive, antisymmetric relations. An x-to-y path A is the *leftmost* x-to-y path in a graph if, for every x-to-y path B, A is left of B. There is not necessarily a leftmost *walk*: suppose $P = Q \circ C$ is the leftmost path where Q is an x-to-y path and C is a c.w. cycle; then $R = P \circ C$ is a walk that is left of P and $R \circ C$ is left of R and so on. However, the following lemma allows us to consider only simple paths when we are considering graphs with no clockwise cycles.

Lemma 4.4. *Let G be a graph with no clockwise cycles. If P is a leftmost walk, then P is a simple path.*

Proof. Assume for a contradiction that P is not a simple path. Let x be a vertex that occurs at least twice on P. Let x_1 be the first occurrence of x on P and let x_2 be the last. Then $C = P[x_1, x_2]$ is a cycle. Since G has no clockwise

cycles, C must be counterclockwise. Let $P' = P[\cdot, x_1] \circ P[x_2, \cdot]$ be the path that is obtained from P by removing C. The circulation

$$\boldsymbol{\delta}(P) - \boldsymbol{\delta}(P') = \boldsymbol{\delta}(C)$$

is counterclockwise, so P is strictly right of P'. Thus P is not leftmost, a contradiction. □

Lemma 4.5. *Every subpath of a leftmost path is a leftmost path.*

Proof. Let P be a leftmost path. Let Q be an x-to-y subpath of P. Suppose there is another x-to-y path $Q' \neq Q$ that is left of Q. Let $P' = P[\cdot, x] \circ Q' \circ P[y, \cdot]$. The circulation

$$\begin{aligned}\boldsymbol{\delta}(P') - \boldsymbol{\delta}(P) &= \boldsymbol{\delta}(P[\cdot, x] \circ Q' \circ P[y, \cdot]) - \boldsymbol{\delta}(P[\cdot, x] \circ Q \circ P[y, \cdot]) \\ &= \boldsymbol{\delta}(Q') - \boldsymbol{\delta}(Q)\end{aligned}$$

is clockwise since Q' is left of Q. So $P' \neq P$ is left of P, a contradiction. □

Theorem 4.6 (Non-Crossing Theorem)**.** *If P and Q are disjoint non-self-crossing x-to-y paths that do not cross each other, then P is either right of or left of Q.*

Proof. Let $C = Q \circ \mathit{rev}(P)$: C is a non-self-crossing cycle. Let G_C be the graph consisting of the edges and vertices of C. Since G_C is a connected graph, each face of G_C has a connected boundary. We show that $\boldsymbol{\delta}(C)$ is either a clockwise or counterclockwise circulation.

We claim that each face of G_C uses either darts of C or darts of $\mathit{rev}(C)$ (but not both). Suppose for a contradiction that f is a face that uses darts of both C and $\mathit{rev}(C)$. Let A and B be maximal subpaths of C such that $A \in \partial f$ and $\mathit{rev}(B) \in \partial f$ and $\mathit{end}(A) = \mathit{start}(\mathit{rev}(B))$. Let A' and B' be the subpaths of C such that $C = A \circ A' \circ B \circ B'$. Since f is a face, C does not cross ∂f and so $A \circ A'$ must leave ∂f from the left (at $\mathit{end}(A)$, by the maximality of A). Likewise, $B \circ B'$ leaves ∂f from the left. Since C is non-self-crossing, $A \circ A'$ does not cross $B \circ B'$. However B' is an $\mathit{end}(B)$-to-$\mathit{start}(A)$ path and A' is a $\mathit{end}(A)$-to-$\mathit{start}(B)$, so A' crosses B', a contradiction, proving the claim. See Figure 7 for an illustration.

Consider the following assignment of numbers to faces.

$$\boldsymbol{\phi}[f] = \begin{cases} -1 & \text{if } \partial f \subset C \\ 0 & \text{if } \mathit{rev}(\partial f) \subset C \end{cases}$$

By the claim, every face is assigned a number.

If $\boldsymbol{\phi}[f_\infty] = 0$, then $\boldsymbol{\phi}$ is a valid potential assignment and corresponds to the circulation $\boldsymbol{\eta} = \boldsymbol{\delta}C$. Then C is clockwise and Q is left of P.

If $\boldsymbol{\phi}[f_\infty] = -1$, then $\boldsymbol{\phi} + \mathbf{1}$ is a valid potential assignment (where $\mathbf{1}$ is the all-ones vector). It follows that C is counterclockwise and Q is right of P. □

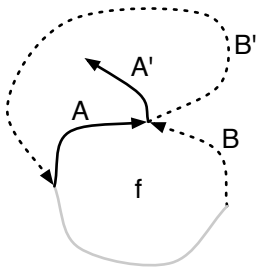


Figure 7: If $A \circ A' \circ B \circ B'$ is a cycle and $A \cup rev(B) \in \partial f$, then A' must cross B'.

4.5 Maximum flows and minimum cuts

We now give the formal statement of the maximum-flow and minimum-cut problems. Given a graph G, a source vertex s, a sink vertex t, and an assignment $c(\cdot)$ of real-valued *capacities* to the arcs of G, the maximum-flow problem is as follows:

$$\begin{aligned} \max \quad & \mathbf{f} \cdot \boldsymbol{\delta}(\delta^+(\{s\})) \\ \text{s.t.} \quad & \mathbf{f} \text{ is a vector in arc-space} && (1) \\ & \mathbf{f} \cdot \boldsymbol{\delta}(\delta^+(\{v\})) = 0, \ \forall v \in V \setminus \{s,t\} && (2) \\ & 0 \leq \mathbf{f}[a] \leq c(a), \ \forall \text{ arcs } a && (3) \end{aligned}$$

Constraint (2) is the conservation constraint: the net flow at every non-source-or-sink vertex is zero. Constraint (3) is the capacity constraint.

A flow assignment $\mathbf{f}$ or st-flow is called feasible if it satisfies these constraints. The goal is to maximize the *value* of the flow, $\mathbf{f} \cdot \boldsymbol{\delta}(\delta^+(\{s\}))$. A flow of value zero is called a *circulation* and is a vector in cycle space.

Given the same input, the minimum st-cut problem is:

$$\begin{aligned} \min \quad & c(\delta^+(S)) \\ \text{s.t.} \quad & s \in S \subseteq V \setminus \{t\} && (4) \end{aligned}$$

A set of vertices S satisfying Constraint (4) is called an st-cut. The *value* of a cut is given by the objective function.

The capacity function $c(\cdot)$ assigns capacities to arcs. We extend it to darts as follows: $c(\langle a, 1\rangle) = c(a)$ and $c(\langle a, -1\rangle) = 0$ for each arc a. That is, a dart in the same direction as the corresponding arc has the same capacity as the arc, and a dart in the opposite direction has capacity zero.

A flow $\mathbf{f}$ assigns values to arcs. We extend it to darts as follows: $\mathbf{f}[\langle a, 1\rangle] = \mathbf{f}[a]$ and $\mathbf{f}[\langle a, -1\rangle] = -\mathbf{f}[a]$.

For any flow $\mathbf{f}$, the *residual capacity* of a dart d, written $c_{\mathbf{f}}(d)$, is $c(d) - \mathbf{f}[d]$.

A dart d is *residual* with respect to $\mathbf{f}$ and c if its residual capacity is positive. Otherwise, d is non-residual. A path is residual if all its darts are residual.

It follows from the Max-Flow Min-Cut Theorem that a feasible st-flow $\mathbf{f}$ is maximum if and only if there is no residual s-to-t path with respect to $\mathbf{f}$ and c. *Augmenting* an st-flow $\mathbf{f}$ along a residual s-to-t path P, means increasing $\mathbf{f}[d]$ by the same amount for each dart d in P. We call this path the *augmenting path*. Suppose that $\mathbf{f}$ is feasible with respect to c. If the amount of the increase is no more than

$$\Delta = \min_{d \in P} c(d) - \mathbf{f}[d],$$

then after augmentation the st-flow $\mathbf{f}$ is still feasible. If the increase is exactly Δ, then we say the augmentation *saturates* the path P. In this case, at least one dart of P becomes *saturated* (i.e., non-residual).

5 The max-flow algorithm

We present an algorithm to find a maximum st-flow in a directed planar graph that runs in $O(n \log n)$ time. The algorithm is a direct generalization of the uppermost-path algorithm. Ford and Fulkerson's algorithm finds the uppermost flow: one in which no flow can be rerouted *above* the existing flow. Our generalization finds the *leftmost* flow (which we define in the next section, and is defined with respect to the infinite face). At the start of the algorithm, we start with a *leftmost flow of value zero*, which is achieved via a preprocessing step equivalent to satisfying Requirement 2 of Weihe's algorithm. The algorithm, which we call MaxFlow, is as follows:

- Designate a face incident to t as f_∞.
- Saturate the clockwise cycles. (LeftmostCirculation)
- While there is a residual s-to-t path, saturate the leftmost such path. (LeftmostFlow)

In Section 5.1, we review an algorithm due to Khuller, Naor, and Klein [26] for carrying out the second step, LeftmostCirculation. This algorithm was previously used by Weihe [36]. In Section 5.2, we discuss the third step, LeftmostFlow. We state a theorem, the Unusability Theorem, that implies that the third step, LeftmostFlow, takes $O(n)$ iterations. We give an implementation in which each iteration takes $O(\log n)$ time. (More precisely, the implementation allows for some degenerate iterations in which zero flow is pushed, but the Unusability Theorem enables us to show that the total number of iterations is nevertheless linear.) In Section 6, we prove the Unusability Theorem.

5.1 LeftmostCirculation

The second step of the MaxFlow algorithm, LeftmostCirculation, is implemented using the following algorithm, due to Khuller, Naor, and Klein [26]. It

computes an assignment of potentials to the faces using single-source shortest-path distances in the dual planar graph, interpreting capacities as distances. (We denote the length of the shortest x-to-y path in G as $dist_G(x, y)$.)

LEFTMOSTCIRCULATION$(G), c(\cdot), f_\infty)$

- Interpret capacities $c(d)$ as lengths of darts in the dual graph G^*.
- Let $\boldsymbol{\phi}[f] = dist_{G^*}(f, f_\infty)$ for every face f.
- Let $\boldsymbol{\eta}[d] = \boldsymbol{\phi}[tail_{G^*}(d)] - \boldsymbol{\phi}[head_{G^*}(d)]$ for every dart d.
- Return $\boldsymbol{\eta}$.

Lemma 5.1 (Khuller et al. [26])**.** *The graph has no clockwise cycle that is residual with respect to the circulation returned by* LEFTMOSTCIRCULATION*.*

Proof. Let C be a clockwise cycle of darts in G and let T be the tree representing the shortest-path distances computed in LEFTMOSTCIRCULATION. There is a path in T to f_∞ from every face enclosed by C, so at least one dart of C is in the shortest-path tree. Let d be such a dart:

$$\begin{aligned} c_\eta(d) &= c(d) - \boldsymbol{\eta}[d] \\ &= c(d) - (dist_{G^*}(tail_{G^*}(d), f_\infty) - dist_{G^*}(head_{G^*}(d), f_\infty)) \\ &= c(d) - c(d) \qquad \text{since } d \text{ is in the shortest-path tree} \\ &= 0 \end{aligned}$$

Since d is not residual, the cycle C is not residual. □

5.2 LEFTMOSTFLOW

Weihe [36] defined a *leftmost* maximum st-flow. We slightly generalize this. We say an st-flow $\mathbf{f}$ (not necessarily maximum) is a *leftmost flow* if the residual graph with respect to $\mathbf{f}$ has no clockwise residual cycles.

Lemma 5.2. *Let* $\mathbf{f}$ *be a leftmost flow and let* P *be the leftmost residual path. The flow* $\mathbf{f}'$ *that results from saturating* P *is a leftmost flow.*

Proof. Assume for a contradiction that $\mathbf{f}'$ is not a leftmost flow. By the definition of leftmost, there must then be a clockwise residual cycle C with respect to $\mathbf{f}'$. Assume w.l.o.g. that C is simple. Since $\mathbf{f}$ was leftmost, C was not residual prior to the augmentation, and so P must have a dart d in common with $rev(C)$.

Let $P' = P[\cdot, tail(d)] \circ C[tail(d), tail(d)] \circ P[tail(d), \cdot]$ where $C[tail(d), tail(d)]$ is the clockwise cycle starting at $tail(d)$. P' is a walk and $\boldsymbol{\delta}(P') - \boldsymbol{\delta}(P) = \boldsymbol{\delta}(P) + \boldsymbol{\delta}(C) - \boldsymbol{\delta}(P) = \boldsymbol{\delta}(C)$ is a clockwise circulation and so P' is left of P. Therefore P was not the leftmost residual path. □

Next we discuss the third step of our MAXFLOW algorithm, LEFTMOSTFLOW: starting with the circulation $\boldsymbol{\eta}$ output by the second step (LEFTMOSTCIRCULATION), repeatedly saturate the leftmost residual s-to-t path until none remains. It is convenient for the analysis to describe LEFTMOSTFLOW as an

algorithm that takes as input a new graph G_0 and new capacity function $c_0(\cdot)$, derived from the original input graph G_{in} and capacity function $c_{\text{in}}(\cdot)$, that satisfies the following preconditions.

Precondition 5.3. *G_0 has no clockwise cycle of arcs.*

Precondition 5.4. *For every arc a in G_0, $c_0(a) > 0$.*

To obtain G_0 and $c_0(\cdot)$ from the original input graph G_{in} and original capacity function $c_{\text{in}}(\cdot)$ in the MaxFlow algorithm, we use the leftmost circulation $\boldsymbol{\eta}$ found in the second step. For each dart of G_{in} that is residual with respect to $\boldsymbol{\eta}$, we include a corresponding arc in G_0, and we assign the residual capacity of the dart as the capacity of the new arc in G_0. Because G_{in} has no cycle of darts that is residual with respect to $\boldsymbol{\eta}$, G_0 satisfies Precondition 5.3. Because each arc in G_0 arises from a residual dart, G_0 satisfies Precondition 5.4.

The algorithm LeftmostFlow finds a maximum st-flow $\mathbf{f}$ in G_0. By adding $\boldsymbol{\eta}$ to $\mathbf{f}$, we obtain a maximum st-flow in the original graph G_{in}.

However, it should be emphasized that this distinction between the input graph G_{in} and G_0 is purely for notational convenience; it enables the analysis to use the term *arc* to refer to a dart that is residual with respect to the leftmost circulation. A simpler and equivalent version of LeftmostFlow would simply continue working with the input graph G_{in} using the circulation found by LeftmostCirculation as the initial flow assignment.

We start with an abstract version of LeftmostFlow.

(Abstract) LeftmostFlow$(G_0, c_0, s, t, f_\infty)$

- Initialize $\mathbf{f} = \mathbf{0}$.
- While there is an s-to-t path that is residual w.r.t. $\mathbf{f}$ and c_0, saturate the leftmost residual s-to-t path, modifying $\mathbf{f}$.
- Return $\mathbf{f}$.

The following invariant of LeftmostFlow follows from Lemma 5.2:

Invariant 5.5. *During the execution of* LeftmostFlow*, G_0 has no clockwise residual cycles with respect to $\mathbf{f}$.*

Corollary 5.6. *During the execution of* LeftmostFlow*, there is no cycle of darts all assigned positive flow.*

Proof. Assume for contradiction that C is a cycle of darts all assigned positive flow. If C is counterclockwise, then $rev(C)$ is residual, contradicting Invariant 5.5. If C is clockwise, then C contradicts Precondition 5.3. □

In Ford and Fulkerson's uppermost-path algorithm for st-planar graphs, once flow is pushed on an arc, flow can never be removed from that arc. For planar graphs that are not st-planar, such a strong property does not hold, as illustrated in Figure 8. However, we prove a weaker property that suffices for the analysis.

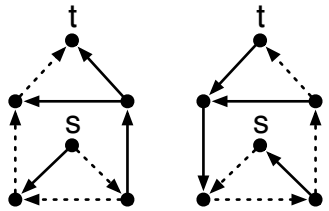


Figure 8: A simple example illustrating that flow can be removed from an arc in LeftmostFlow, even in the case of unit capacities. On the left, the leftmost residual path (dotted) pushes flow along the bottom arc. On the right is the resulting residual graph. The leftmost residual path (dotted) removes flow from the bottom arc.

Theorem 5.7 (Unusability Theorem)**.** *Consider the algorithm* LeftmostFlow*. Suppose an arc a is augmented and some time later $rev(a)$ is augmented. Then arc a cannot be augmented again.*

We informally explain how this theorem enables us to bound the number of iterations; a more formal explanation appears later in this section. If an arc a is saturated, the algorithm must augment $rev(a)$ before augmenting a again. The theorem therefore shows that each arc is saturated at most once. If $rev(a)$ is saturated, the algorithm must previously have augmented a; after $rev(a)$ is saturated, a itself must be augmented before $rev(a)$ can be saturated a second time. The theorem therefore shows that the reverse of each arc is saturated at most once. Each iteration of the abstract version of LeftmostFlow saturates some arc or the reverse of some arc, so the number of iterations is at most twice the number of arcs.

In order to achieve $O(\log n)$ time per iteration, we give an implementation of LeftmostFlow in which some iterations do not actually push any flow, but we can nevertheless use the Unusability Theorem to bound the number of iterations by thrice the number of arcs.

The algorithm is given in Table 2). It maintains a spanning tree T of the graph rooted at the sink t and the corresponding dual spanning tree T^* rooted at the infinite face f_∞.

The tree is an undirected structure, so we modify it by ejecting or inserting undirected edges, but as shorthand we speak of ejecting or inserting darts.

Right-first search [31] in Step 2 constructs a tree T spanning every vertex v that can reach t in G_0, such that the path $T[v]$ is the leftmost directed v-to-t path in G_0. The primal tree T is represented using a *dynamic-tree* data structure [1, 2, 12, 33, 35], enabling Steps 6, 7, and 11 to run in amortized $O(\log n)$ time. The dual tree T^* is represented by an Euler-tour tree data structure [20], so Steps 8, 9 and 10 can also be implemented in amortized $O(\log n)$ time.

An iteration of Step 5 is a *pivot step* and is illustrated in Figure 9. To show that LeftmostFlow(G_0, c, s, t) takes $O(m \log n)$ time, we show that there are at most $3m$ pivot steps (Theorem 5.15). It therefore follows that the algorithm

(Implementation) LEFTMOSTFLOW(G_0, c_0, s, t)

1. Initialize $\mathbf{f} = \mathbf{0}$.
2. Initialize T to be the right-first search tree searching backwards from t. (Every arc in T is directed towards t.)
3. Let G be the graph obtained from G_0 by deleting all vertices not in T.
4. Initialize T^* to consist of the edges of G^* that are not in T.
5. Repeat:
    6. If $T[s]$ is residual, saturate $T[s]$, modifying $\mathbf{f}$.
    7. Let d be the last non-residual dart in $T[s]$.
    8. If $tail_{G^*}(d)$ is a descendent in T^* of $head_{G^*}(d)$, return $\mathbf{f}$.
    9. Let e be the parent dart in T^* of $head_{G^*}(d)$.
    10. Eject e from T^* and insert d into T^*.
    11. Eject d from T and insert e into T.

Table 2: A network-simplex type implementation of the LEFTMOSTFLOW algorithm.

runs in $O(m \log n)$ time.

First we show that the algorithm does maintain spanning trees of G and G^*.

Invariant 5.8. *T is a spanning tree of G and T^* is a spanning tree of G^*.*

Proof that the algorithm maintains the invariant. Initially, Step 2 establishes that T is a spanning tree of G and Step 4 establishes that T^* is a spanning tree of G^*, by the Interdigitating Spanning Trees Theorem. Ejecting e from T^* in Step 10 breaks T^* into two connected component, one consisting of the descendents of $head_{G^*}(d)$ in T^* and one consisting of the non-descendents. At the time Step 10 is about to be executed, the condition in Step 8 is false, so $tail_{G^*}(d)$ is not a descendent of $head_{G^*}(d)$ in T^*. It follows that in T^* inserting d joins the two connected components, so T^* is a spanning tree of G^* at the end of Step 10. By the Interdigitating Spanning Trees Theorem, therefore, T is a tree of G after Step 11. □

Note that Step 11 can result in reversing parent and child in some edges. Specifically, as illustrated in Figure 9, the path in T between $tail_G(d)$ and $tail_G(e)$ is reversed. However, this is within the scope of the dynamic-tree data structure.

Invariant 5.9. *Let e be an edge of T^*, and let d be the corresponding dart that is oriented away from f_∞. Then d is non-residual.*

Proof that the algorithm maintains the invariant. First we show that the invariant holds initially. T^* is composed of edges not in T. Let a be any arc not in T. By construction of T, the path of arcs $a \circ T[head_G(a)]$ is right of $T[tail_G(a)]$.

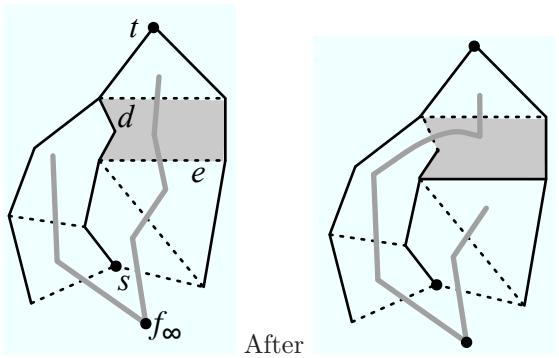


Figure 9: The edges of T are solid, non-tree edges are dashed. The root of T is the sink t. T^* is represented by light edges, and its root is the infinite face f_∞. In an iteration of LeftmostFlow, the s-to-t path is saturated and d is the rootmost non-residual edge. The shaded face is $head_{G^*}(d)$. This face's parent dart e in T^* is ejected from T^* and inserted into T, and $rev(d)$ is inserted into T^* and becomes the new parent dart. The edge that immediately precedes d in the s-to-t path gets reversed: the parent endpoint becomes the child endpoint and vice versa.

Let $C = a \circ T[head_G(a), tail_G(a)]$. Then C is a simple c.c.w. cycle. The face to the left of a is enclosed by C and the face to the right is not. Let S be the set of faces enclosed by C. In G^*, a is directed out of S (i.e. $a \in \delta^+_{G^*}(S)$). Since a is the only arc in C that is not in T and therefore is in T^*, and since f_∞ is not enclosed by C, $\langle a, 1\rangle$ is directed towards f_∞ in T^* and $\langle a, -1\rangle$ is oriented away from f_∞. Since the reverses of arcs have zero capacity, the invariant holds initially.

See Figure 9 for an illustration of the next argument. Note that, in each nonterminating pivot step, $tail_{G^*}(d)$ is not a descendent in T^* of $head_{G^*}(d)$. Dart e, the parent of $head_{G^*}(d)$, is removed from T^*. The component of $T^* - \{e\}$ that contains f_∞ contains $tail_{G^*}(d)$ and not $head_{G^*}(d)$, so when d is inserted into T^*, d is oriented away from f_∞. Since d was saturated in Step 6, d is non-residual.

Let c be a dart that remains in T^* during a pivot. The residual capacity of c does not change and the orientation of c in T^* does not change. Therefore the invariant holds. □

We say that a dart d is a *non-tree dart* if the corresponding edge is not in T.

Lemma 5.10. *There is no clockwise simple cycle whose non-tree darts are all residual.*

Proof. Suppose for a contradiction that C was such a cycle. Let S be the set of non-tree darts in C. By Invariant 5.8, for every dart $d \in S$, the tree T^* contains the edge corresponding to d. Since every dart in S is residual,

Invariant 5.9 implies that in T^* the darts of S are oriented towards the root f_∞. Since C is clockwise, $head_{G^*}(d)$ is enclosed by C for every dart d in C. Since T is a tree, C contains at least one non-tree dart d. The directed path $T^*[head_{G^*}(d)]$ is completely enclosed by C, implying that C also encloses $f_\infty = end(T^*[head_{G^*}(d)])$, a contradiction. □

We show in the next two corollaries that the network-simplex version of LeftmostFlow implements the abstract version. Corollary 5.11 shows that every augmentation is along the leftmost residual s-to-t path and Corollary 5.12 shows that the algorithm does not terminate until there is no s-to-t residual path.

Corollary 5.11. *For every vertex v, there is no residual path strictly left of $T[v]$.*

Proof. Suppose for a contradiction that there is a residual path strictly left of $T[v]$. Then the leftmost residual v-to-t path P must be strictly left of $T[v]$. Let Q be a subpath of P such that the endpoints of Q are on $T[v]$ but Q and $rev(Q)$ are both edge disjoint from $T[v]$. Since P is leftmost, by Lemma 4.5, Q is left of $T[end(Q), start(Q)]$, so $Q \circ rev(T[end(Q), start(Q)])$ is a simple c.w. cycle whose non-tree darts are residual, contradicting Lemma 5.10. □

Corollary 5.12. *The st-flow $\mathbf{f}$ returned by the algorithm is maximum.*

Proof. Refer to Figure 10. When the algorithm terminates in Step 7, $tail_{G^*}(d)$ is a descendent in T^* of $head_{G^*}(d)$. Let C be the simple cycle $d \circ T^*[head_{G^*}(d), tail_{G^*}(d)]$ in the dual. In the primal G, the darts of C form a directed cut $\delta_G^+(S)$. Every dart in C except d is a non-tree dart, so the $head_G(d)$-to-t path in T does not use any dart in C or the reverse of any dart in C. Since t is on the infinite face, C does not enclose t and so S does not contain t. Likewise the s-to-$tail_G(d)$ path in T does not use any dart in C or the reverse of any dart in C. Since d crosses C, S contains s. Since every dart comprising the cut is non-residual, there is no residual s-to-t path. By the Max-Flow Min-Cut Theorem, the flow is maximum. □

We now show that there are at most $3m$ pivot steps in the LeftmostFlow algorithm. Let d be a dart. We have the following facts with regards to the LeftmostFlow algorithm:

Fact 1. If d is residual at time i and non-residual at time j $(i < j)$, there was an augmentation including d at some time between i and j.

Fact 2. If d is non-residual at time i and residual at time j $(i < j)$, there was an augmentation including $rev(d)$ at some time between i and j.

Fact 3. When e is inserted into T, e is residual. (Just before e was inserted into T, e was a parent dart in T^*. By Invariant 5.9, $rev(e)$ is non-residual. Precondition 5.4 implies that e is residual.)

Fact 4. When d ejected from T, d is non-residual. (This holds by choice of d in Step 7.)

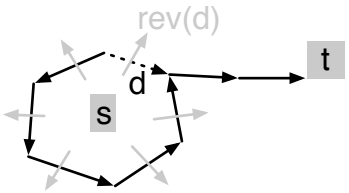


Figure 10: An illustration of Corollary 5.12. Darts of the dual tree are dark, with darts of T^* solid. In the dual, s and t are faces, shown shaded. Upon termination, $tail_{G^*}(d)$ is a descendent in T^* of $head_{G^*}(d)$. Since $rev(d)$ is non-residual and the reverses of dual tree darts are non-residual, the cycle shown is a saturated cut separating s from t. The darts of this cut (in the primal) are light.

Claim 5.13. *A dart $\langle a, 1\rangle$ where a is an arc of G_0 is ejected at most once.*

Proof. Let a be an arc and let $d = \langle a, 1\rangle$. Suppose for a contradiction that a is ejected at time i_1 and at time i_2 ($i_1 < i_2$).

To be ejected at time i_1, d must be non-residual by Fact 4. Fact 1 implies that there was an augmentation including d at some time k_0 where $0 < k_0 < i_1$.

To be ejected at time i_2, d must have been inserted at some time j_1 where $i_1 < j_1 < i_2$. At time j_1, d is residual by Fact 3. By Fact 2, there was an augmentation including $rev(d)$ at some time k_1 where $i_1 < k_1 < j_1$.

Since there was an augmentation including d at time k_0 and there was an augmentation including $rev(d)$ at time $k_1 > k_0$, d cannot be augmented after time k_1 by the Unusability Theorem.

Finally, to be ejected at time i_2, d must be non-residual by Fact 4. By Fact 2, there was an augmentation including d at some time k_2 where $j_1 < k_2 < i_2$. But d cannot be augmented after time k_1. This is a contradiction. □

Corollary 5.14. *A dart $\langle a, -1\rangle$ where a is an arc of G_0 is ejected at most twice.*

Proof. Let a be an arc and let $d = \langle a, -1\rangle$.

Suppose d is ejected at times i_1 and i_2. Then d must be inserted at time $i_1 < j_1 < i_2$. By Fact 4, d is non-residual at time i_1 and by Fact 3, d is residual at time j_1. By Fact 2, $rev(d)$ must be part of an augmentation at some time k_1 where $i_1 < k_1 < j_1$.

Likewise, by Fact 4, d is non-residual at time i_2 and by Fact 1 d must be augmented at time k_2 where $j_1 < k_2 < i_2$.

At time i_2, d is out of the tree and non-residual. Since $rev(d)$ cannot be augmented after time k_2 by Claim 5.13, d can never become residual again and so cannot be inserted or ejected again. □

As a consequence of the above, we have the following theorem:

Theorem 5.15. *There are at most* $3m$ *pivot steps in the* LeftmostFlow *algorithm.*

6 Unusability Theorem

In this section, we prove the Unusability Theorem. The structure of the proof is as follows. We show that if an arc a is augmented and later $rev(a)$ is augmented, then a structure in the residual graph arises called an *obstruction* (Lemma 6.11). We show that this structure persists under leftmost augmentations (Lemma 6.12). We also prove (Lemma 6.10) that the existence of the obstruction ensures that no leftmost residual path includes arc a, which shows that a is never augmented again.

The idea of the proof is as follows. We assume for a contradiction that the leftmost residual path A does include a. We use a suffix of the s-to-$tail(a)$ subpath of A together with paths comprising the obstruction to construct a cycle C such that the arc a is completely enclosed by C, as shown in Figure 16(b), and we show that the $head(a)$-to-s subpath of A cannot escape from this cycle (escaping would imply that an invariant failed to hold).

We make use of Preconditions 5.3 (no c.w. cycles) and 5.4 (every arc is initially residual). We will use the term *arc* to refer to an arc of G_0 (and to the corresponding dart), and use *anti-arc* to refer to a dart whose reverse is an arc.

Before commencing the proof, we establish some properties of leftmost residual paths and walks. Recall that since the graphs we consider have no clockwise residual cycles (Invariant 5.5), the leftmost walk is a simple path (Lemma 4.4).

Lemma 6.1 (Prohibited augmentations)**.** *The following situations are not permitted if A is a leftmost augmentation and the given vertex indices are well-defined:*

1. *$A[x, y]$ is right of a residual walk $R[x, y]$.*
2. *$A[x, y]$ makes a clockwise cycle with residual walk $R[y, x]$.*
3. *A has a dart that enters a t-to-s residual walk R from the right.*

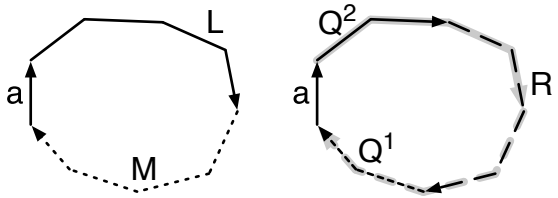
Proof. We prove each part separately.

1. $A[s, x] \circ R[x, y] \circ A[y, t]$ is left of A. This contradicts the requirement that A is leftmost residual walk.
2. This contradicts Invariant 5.5.
3. Suppose uv is a dart of a leftmost augmentation path and suppose uv enters a t-to-s residual walk R from the right. As such, $uv \notin R$ and $rev(uv) \notin R$. $A[s, u]$ must intersect R at some vertex: let x be the last intersection of $A[s, u]$ with R (possibly $x = s$). If $x \in R(v, s]$, then $A[x, v] \circ R[v, x]$ is clockwise, contradicting Invariant 5.5. If $x \in R(t, v)$ then $R[x, v]$ is a residual walk that is left of $A[x, v]$, which is a prohibited augmentation of the first kind. □

Now we define what it means to be unusable. Unusability is given by a structure in the residual graph called an *obstruction*.

Definition 6.2 (Unusable Arc)**.** *An* obstruction *is a clockwise non-self-crossing cycle $L \circ M$ where L is residual and M consists entirely of arcs. We say it is an obstruction* for *an arc a if $\langle a, 1\rangle$ is the first dart of L. We say an arc a is* unusable *if there is an obstruction for a.*

We give a second, equivalent representation for an obstruction which will be useful in proving the Unusability Theorem. Both are illustrated in Figure 11.



(a) (b)

Figure 11: (a) An obstruction for arc a as given by Definition 6.2. (b) The obstruction for arc a as given by Lemma 6.3 with the obstruction from (a) shaded in the background. L, Q^2 and R are residual; M, Q^1 and Q^2 consist of arcs; a is the first arc of L and Q^2.

Lemma 6.3. *A clockwise non-self-crossing cycle C satisfies Definition 6.2 iff it can be written as $Q^1 \circ Q^2 \circ R$ where*

1. *$Q^1 \circ Q^2$ consists entirely of arcs,*
2. *$Q^2 \circ R$ is residual,*
3. *a is the first dart of Q^2,*
4. *there is flow through the vertex $start(R)$, and*
5. *there is flow through the vertex $end(R)$.*

Proof. The "if" direction is trivial. To prove the "only if" direction, let $L \circ M$ be an obstruction for a. Since G_0 has no clockwise cycle of arcs, $L \circ M$ cannot consist entirely of arcs. Let b be the first anti-arc of L. By Invariant 5.5, $L \circ M$ cannot consist entirely of residual darts and so M cannot consist entirely of residual darts. Let c be the first non-residual dart of M.

Let $Q^1 = M[tail(c), \cdot]$, let $Q^2 = L[\cdot, tail(b)]$, and let $R = L[tail(b), \cdot] \circ M[\cdot, tail(c)]$. By choice of b, $L[\cdot, tail(b)]$ consists entirely of arcs, so property 1 holds. By choice of c, $M[\cdot, tail(c)]$ is residual, so property 2 holds. Since a is the first dart of L, property 3 holds. Since b is a residual anti-arc, $rev(b)$ carries flow, so property 4 holds. Since c is a non-residual arc, by Precondition 5.4 it carries flow, so property 5 holds. □

Definition 6.4. *For an unusable arc a, let Δ_a denote the obstruction for a that encloses the minimum number of faces (breaking ties arbitrarily). Write Δ_a as $Q_a^1 \circ Q_a^2 \circ R_a$, and let Q_a denote $Q_a^1 \circ Q_a^2$.*

Considering a *minimally enclosing* obstruction simplifies the proofs because it allows us to rule out the existence of certain paths that cross the obstruction. Let C be a non-self-crossing cycle, and let P be a path whose start and end are vertices of C. If P has at least one dart, we say P *crosscuts* C if every dart of P is strictly enclosed by C. If P has no darts (i.e. $start(P) = end(P)$), we say P crosscuts C if $start(P)$ occurs more than once in C. Note that in either case the path P splits C into two cycles, e.g. containing strictly fewer faces.

A non-trivial path P is a *flow path* if every dart of P is assigned a positive flow value. A trivial path P (i.e. having no darts) is a flow path if there is a dart incident to $start(P)$ that is assigned a positive flow value. Note that a dart assigned a positive flow value must correspond to an arc, i.e. cannot be an anti-arc.

Property 6.5. *Suppose a is unusable. There is no residual path crosscutting Δ_a from a vertex in $Q_a^2(\cdot,\cdot]$ to a vertex in Q_a^1.*

Proof. Assume for a contradiction that W is such a residual path. Then $W \circ Q_a^1[end(W),\cdot] \circ Q_a^2[\cdot, start(W)]$ is an obstruction for arc a that encloses fewer faces than Δ_a does. That is, W can be used to replace R_a in the obstruction. □

Property 6.6. *Suppose a is unusable. Q_a^2 belongs to a t-to-s residual path.*

Proof. Since Δ_a is a clockwise cycle, it cannot be residual, so Q_a^1 cannot be residual. Let b be the last non-residual dart of Q_a^1. Since Q_a^1 contains only arcs, b carries flow and this flow must be routed to t. Let F_t be any $head(b)$-to-t flow path and let F_s be any s-to-$start(R_a)$ flow path. Since the reverse of a flow path is residual, the path $rev(F_t) \circ Q_a^1[head(b),\cdot] \circ Q_a^2 \circ rev(F_s)$ is a residual t-to-s path. □

Property 6.7. *There are no flow paths that crosscut Δ_a.*

Proof. Assume for contradiction that F is such a flow path, and assume without loss of generality that F is simple. Let $\alpha = start(F)$ and $\beta = end(F)$. Then $C_1 = \Delta_a[\alpha,\beta] \circ rev(F)$ and $C_2 = F \circ \Delta_a[\beta,\alpha]$ are clockwise non-self-crossing cycles, each enclosing fewer faces than Δ_a. See Figure 12.

We will refer to the following:

Argument 1 Note that $rev(F)$ is residual. If $\Delta_a[\alpha,\beta]$ were residual then C_1 would be a residual clockwise cycle, contradicting Invariant 5.5. Since all non-residual darts of Δ_a are in Q_a^1, we infer that $\Delta_a[\alpha,\beta]$ must include at least one dart of Q_a^1.

Argument 2 Note that F consists entirely of arcs. If $\Delta_a[\beta,\alpha]$ consisted entirely of arcs then C_2 would be a clockwise cycle of arcs in G_0, contradicting Precondition 5.3. Since all anti-arcs of Δ_a are in R_a, we infer that $\Delta_a[\beta,\alpha]$ must include at least one dart of R_a.

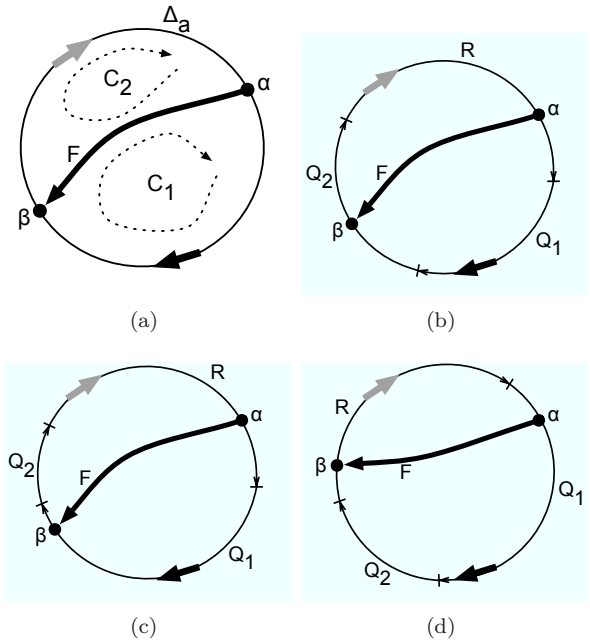


Figure 12: Property 6.7 (a) An illustration of the cycles C_1 and C_2 for the the proof of Property 6.7. By Argument 1, there is an arc (bold) of Q_a^1 in $\Delta_a[\alpha, \beta]$. By Argument 2, there is an arc (grey) of R_a in $\Delta_a[\beta, \alpha]$. (b) Case 1, first sub-case. C_1 is a smaller obstruction. (c) Case 1, second sub-case. C_2 is a smaller obstruction. (d) Case 2. C_1 is a smaller obstruction.

There are two cases to consider:

Case 1 *start*(R_a) is a vertex of $\Delta_a[\beta, \alpha]$: By Argument 1, Q_a^1 is not a subpath of $\Delta_a[\beta, \alpha]$. If a is in $\Delta_a[\alpha, \beta]$ then C_1 is an obstruction enclosing fewer faces than Δ_a (Figure 12b). If a is in $\Delta_a[\beta, \alpha]$ then C_2 is an obstruction enclosing fewer faces than Δ_a (Figure 12c).

Case 2 *start*(R_a) is a vertex of $\Delta_a(\alpha, \beta)$: By Argument 2, R_a is not a subpath of $\Delta_a[\alpha, \beta]$, so *end*(R_a) is outside $\Delta_a[\alpha, \beta]$. By Argument 1, Q_a^1 is not a subpath of $\Delta_a[\beta, \alpha]$, so α is a vertex of Q_a^1. Therefore the first arc of Q_a^2, which is a, is in $\Delta_a[\alpha, \beta]$, so C_1 is an obstruction enclosing fewer faces than Δ_a (Figure 12d).

Each case contradicts the minimality condition of Δ_a. □

Corollary 6.8. *If Δ_a strictly encloses a flow-carrying dart d, then Δ_a strictly encloses the source s and every s-to-head(d) flow path.*

Proof. Suppose for a contradiction that there is a flow-carrying dart d strictly enclosed by Δ_a and an s-to-*head*(d) flow path P containing a dart e that is not

strictly enclosed by Δ_a. Let Q be a $head(d)$-to-t flow path. The path $P \circ Q$ contains a subpath that starts at a vertex ($head(e)$) not strictly enclosed by Δ_a, goes through a dart (d) strictly enclosed by Δ_a, and ends at a vertex (t) not strictly enclosed by Δ_a. Such a flow path violates Property 6.7. □

For an unusable arc a, there is a $start(R_a)$-to-t flow path and an s-to-$end(R_a)$ flow path by Parts 4 and 5 of Lemma 6.3.

Corollary 6.9. *For an unusable arc a, any $start(R_a)$-to-t flow path does not intersect any s-to-$end(R_a)$ flow path.*

Proof. Let F_t be any $start(R_a)$-to-t flow path and let F_s be any s-to-$end(R_a)$ flow path. By Corollary 5.6, each of these paths is simple. Suppose for a contradiction that F_t and F_s share a vertex. Let w be the first such vertex in F_t. See Figure 13. Let F'_s be the maximal suffix of F_s that is not strictly enclosed by Δ_a. By Corollary 6.8, F'_s is the only part of F_s that is not strictly enclosed by Δ_a. Since F_t ends at a vertex that is not strictly enclosed by Δ_a, Property 6.7 implies that no arc of F_t is strictly enclosed by Δ_a, so w must be a vertex of F'_s.

Let $F = F_t[start(R_a), w] \circ F_s[w, end(R_a)]$. F is a $start(R_a)$-to-$end(R_a)$ flow path that is not internal to Δ_a. By the Non-Crossing Theorem, F is either right of or left of R_a. There are two cases.

Case 1 If F is right of R_a, $R_a \circ rev(F)$ is a clockwise residual cycle, contradicting Invariant 5.5.

Case 2 If F is left of R_a, F is also left of $rev(Q_a)$ by transitivity. Hence $F \circ Q_a$ is a clockwise cycle in G_0, a contradiction. This case is illustrated in Figure 13. □

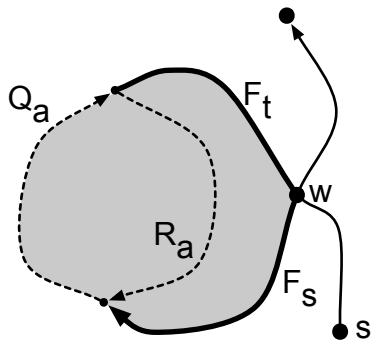


Figure 13: If flows to and from Δ_a (dashed), F_s and F_t, share a vertex w, then we can construct from them a $start(R_a)$-to-$end(R_a)$ flow path F (bold). The shaded area is bounded by a clockwise cycle of arcs.

Now we have all the tools needed to prove the Unusability Theorem. We prove it in three parts. First we show that an unusable arc cannot belong to a leftmost residual path (Lemma 6.10). Next we show that if an arc a satisfies the condition of the Unusability Theorem, there is an obstruction for a in the residual graph (Lemma 6.11). Finally we show that obstructions persist under leftmost augmentations (Lemma 6.12). The Unusability Theorem follows.

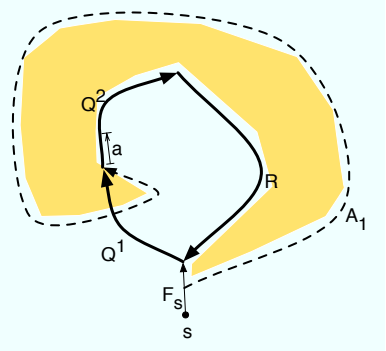


Figure 14: Lemma 6.10, construction of C_1. The obstruction Δ_a is shown in bold. It consists of paths R, Q^1, and Q^2, where a is the first arc of Q^2. The s-to-$end(R_a)$ flow path F_s is indicated by a solid line. The dashed curve A_1 denotes a subpath of a leftmost residual path A that is assumed to include the arc a. The interior of the cycle C_1 is shaded.

Lemma 6.10 (Unusable Arc Consequence)**.** *A leftmost augmenting path contains no unusable arcs.*

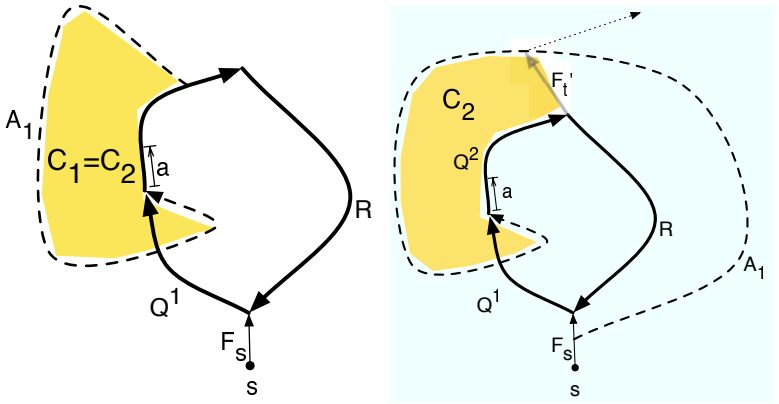
Proof. Let A be the leftmost augmenting path, and assume for a contradiction that it goes through an unusable arc a.

The goal is to first construct a non-self-crossing cycle C that strictly encloses a and does not enclose t. $A[tail(a), \cdot]$ must therefore cross C. We will show that this results in a contradiction.

Consider the flow assignment just before augmenting. Refer to Figure 14. By the definition of Δ_a, there is an s-to-$end(R_a)$ flow path. Let F_s be any such path and let $P_1 = Q^2_a \circ R_a \circ rev(F_s)$. P_1 is a residual $tail(a)$-to-s path. Let A_1 be the maximal suffix of $A[s, tail(a)]$ that does not cross P_1. Let $P'_1 = P_1[\cdot, start(A_1)]$. Let $C_1 = A_1 \circ P'_1$. Then C_1 is a residual cycle and since there are no c.w. residual cycles, it is c.c.w. By construction, C_1 is non-self-crossing.

We next define another c.c.w. non-self-crossing cycle, C_2. Refer to Figure 15. If P'_1 does not include $start(R_a)$, define $C_2 = C_1$. See Figure 15(a). Note that in this case P'_1 is a subpath of Q^2_a.

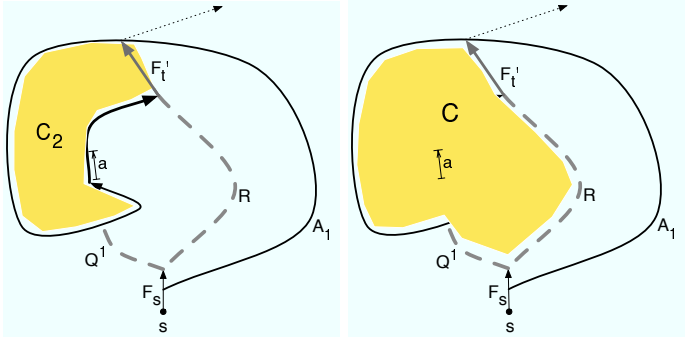
Otherwise, we proceed as follows. See Figure 15(b). By the definition of Δ_a, there is a $start(R_a)$-to-t flow path. Let F_t be any such path and let F'_t be the maximal prefix of F_t that is enclosed by C_1 (possibly the empty path). By Corollary 6.9, F_t and F_s do not share any vertices and by Corollary 6.8, no



(a) (b)

Figure 15: Lemma 6.10, the construction of C_2. In both cases, the interior of C_2 is shaded. (a) An example where the subpath A_1 starts in Q^2. In this case P_1' does not include $start(R_a)$, so C_2 is defined to be C_1. (b) The case where P_1' includes $start(R_a)$. In addition to the paths in Figure 14, this figure illustrates the $start(R_a)$-to-t flow path F_t. The prefix F_t' that is enclosed by C_1 is indicated in gray.

part of F_t is enclosed by Δ_a. We conclude that $end(F_t')$ is in A_1. We define $C_2 = F_t' \circ A_1[end(F_t'), \cdot] \circ P_1[\cdot, start(F_t')]$. Note that $P_1[\cdot, start(F_t')] = Q_a^2$. By definition, C_2 is non-self-crossing.



(a) (b)

Figure 16: Lemma 6.10. (a) The path P_2, indicated by the dashed gray curve, is the maximal prefix of $R \circ Q^1$ not enclosed by C_1. The interior of the cycle C_2 is shaded. (b) The cycle C is obtained by combining the cycle C_2 with the path $rev(P_2)$. Its interior is shaded. Note that C strictly encloses the arc a. The $head(a)$-to-t subpath of the augmentation path A must escape from C at some point.

Note that, in both cases, R_a is not enclosed by C_2. Refer to Figure 16. Let P_2 be the maximal subpath of $Q_a^2 \circ R_a \circ Q_a^1$ that is not enclosed by C_2. By applying the Composition Lemma (Lemma 4.3) to C_2 and $rev(P_2)$, we get a non-self-crossing cycle, C whose boundary is composed of subpaths of F_t, A, $rev(Q_a^1)$, $rev(R_a)$, and possibly $rev(Q_a^2)$. Further, C is c.c.w. and strictly encloses a.

Let A_3 denote the maximal prefix of $A[head(a), \cdot]$ that does not cross C. The three cases are illustrated in Figure 17.

Case 1 $end(A_3) \in Q_a^2 \circ R_a$. Let $P_3 = Q_a^2 \circ R_a$: $P_3[start(A_3), end(A_3)]$ is a boundary of C and since A_3 is enclosed by C, A_3 is right of $P_3[start(A_3), end(A_3)]$, violating Part 1 of Lemma 6.1.

Case 2 $end(A_3) \in rev(F_t')$. Since $rev(F_t')$ is a subpath of a t-to-s residual path, this case contradicts Part 3 of Lemma 6.1.

Case 3 $end(A_3) \in Q_a^1$. Let A_4 be the maximal suffix of A_3 that is internal to Δ_a. The only boundary vertices of Δ_a that are not boundary vertices of C are the vertices of Q_a^2 so $start(A_4)$ must be a vertex of Q_a^2. This case therefore contradicts Property 6.5. □

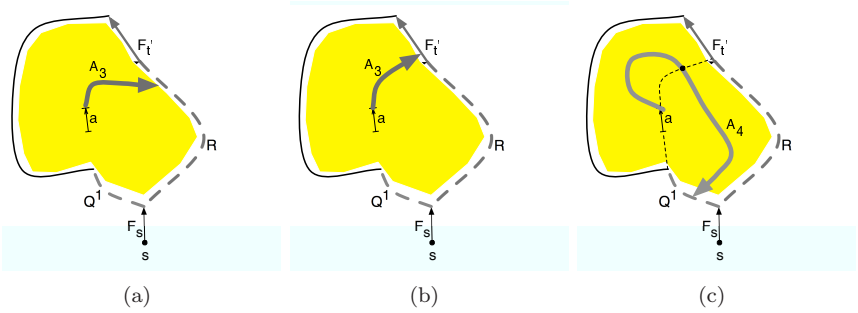


Figure 17: Lemma 6.10. (a) Case 1: The augmenting path leaves C via a vertex of $Q^2 \circ R$. (b) Case 2: The augmenting path leaves C via a vertex of F_t'. (c) Case 3: The augmenting path leaves C via a vertex of Q^1. In this case, a subpath of the augmenting path that is enclosed by Δ_a goes from a vertex of Q^2 to a vertex of Q^1.

Lemma 6.11 (Unusable Arc Creation)**.** *If augmentation A uses arc a in the reverse direction, a will be unusable after augmentation A.*

Proof. See Figure 18. Let a be an arc and let A be the leftmost residual s-to-t path. Suppose d is a dart in A where $d = rev(a)$. Since d is residual, a must carry flow. Let F be any s-to-$tail(a)$ flow path. Let x be the last vertex of $A[\cdot, tail(a)]$ that is in F. Let $L = rev(A[x, tail(a)])$ and let $M = F[x, tail(a)]$.

Both L and M are simple and by the choice of x, L does not cross M. L is residual after augmentation and a is the first dart of L. M consists entirely of arcs. Since $rev(M)$ is residual before augmentation, $A[x, tail(a)]$ must make a c.c.w. cycle with it by Part 2 of Lemma 6.1. Therefore $M \circ L$ is a c.w. non-self crossing cycle, and hence an obstruction for a. □

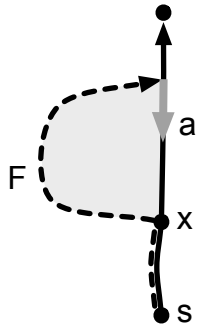


Figure 18: Lemma 6.11. The creation of an obstruction (whose interior is shaded) from the flow path (dotted) through a (grey) and the augmentation path through $rev(a)$ (solid).

Lemma 6.12 (Unusable Arc Persistence)**.** *Once an arc becomes unusable, it remains unusable.*

Proof. Suppose a is an unusable arc at some point in time, and let A be the leftmost residual s-to-t path at that time. The obstruction Δ_a for a remains an obstruction after augmentation along A unless the augmentation renders some dart of $Q_a^2 \circ R_a$ non-residual. Since every dart of Q_a^2 is an unusable arc, Lemma 6.10 implies that the augmentation cannot contain a dart of Q_a^2. We may therefore assume that A and R_a share a dart.

Let b be the first dart of R_a that is in A. Let A_1 be the maximal suffix of $A[\cdot, head(b)]$ that is not strictly enclosed by Δ_a. Since A cannot enter R_a from the right by Part 2 of Lemma 6.1 and since the left of R_a is not enclosed by Δ_a, A_1 is not a trivial path.

Case 1: $A_1 \neq A[\cdot, head(b)]$. This case is illustrated in Figure 19. Let c be the dart of A just preceding A_1. By construction of A_1, the dart c is strictly enclosed by Δ_a. Both Q_a^2 and R_a belong to t-to-s residual paths (by Property 6.6 and by consequence of Lemma 6.3, respectively). Since c is strictly enclosed by Δ_a and so enters Δ_a from the right, $head(c)$ cannot belong to either Q_a^2 or R_a by Part 3 of Lemma 6.1. Thus $head(c)$ is on Q_a^1. Write $\Delta_a = P \circ P'$ where P is a $head(c)$-to-$head(b)$ path that contains b. (That is, $P = Q_a^1[head(c), \cdot] \circ Q_a^2 \circ R_a[\cdot, head(b)]$.) Since A_1 does not cross P, A_1 is either left of or right of P.

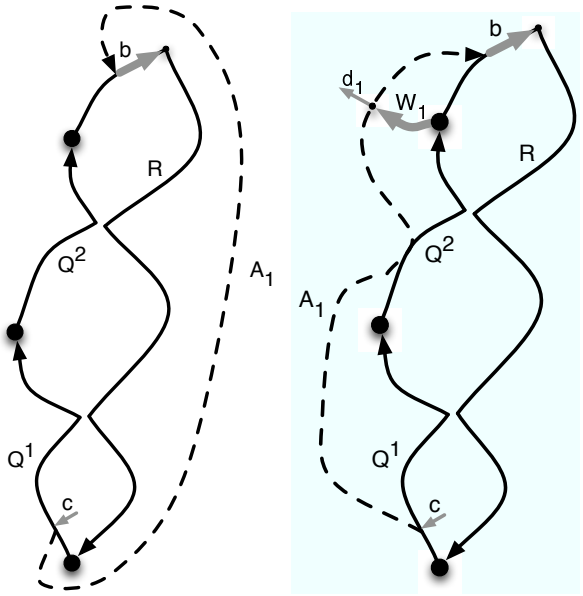
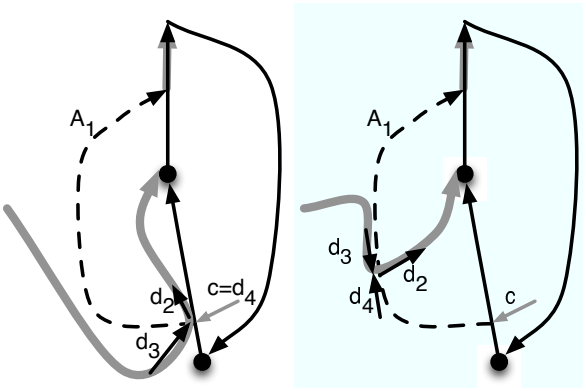


Figure 19: Lemma 6.12, Case 1. The obstruction Δ_a is depicted by the solid lines. The path P, on the left, consists of Q^1, Q^2, and the prefix of R ending with dart b. The remainder of R, which is denoted P', appears on the right. The subpath A_1 of the augmenting path is represented by the dashed line. (a) A_1 is right of P. In this case, combining A_1 with part of P yields a new obstruction for a. (b) A_1 is left of P. The flow path W_1 is shown as a thick gray arrow. The proof shows that this path ends on a vertex of A_1 and is followed in F_t by at least one dart.

First suppose A_1 is right of P. This is the situation depicted in Figure 19(a). Since $rev(A_1)$ is residual after augmentation, $rev(A_1[\cdot, tail(b)]) \circ P[head(c), tail(b)]$ is an obstruction for a after augmentation, proving the lemma.

Suppose therefore that A_1 is left of P. This is the situation depicted in Figure 19(b). Let F_t be a $start(R_a)$-to-t flow path, and let W_1 be the maximal prefix of F_t consisting of darts enclosed by the cycle $A_1 \circ rev(P)$. Let W_2 be the maximal prefix of $F_t[end(W_1), \cdot]$ that consists only of darts strictly enclosed by Δ_a. We claim that W_1 ends on a vertex of the cycle $A_1 \circ P'$, and W_2 contains no darts.

If $W_1 = F_t$ then W_2 contains no darts, and, since t is incident to the infinite face, which is not enclosed by $A_1 \circ P'$, $end(W_1)$ is a vertex of $A_1 \circ P'$, proving the claim. Assume therefore that $W_1 \neq F_t$. If W_2 were nontrivial then W_2 would cross Δ_a, contradicting Property 6.7. Therefore W_2 contains no darts, and W_1 is immediately followed in F_t by a dart d such that d is not enclosed by $A_1 \circ rev(P)$ and not strictly enclosed by $\Delta_a = P \circ P'$. Since every dart of W_1 is



(a) (b)

Figure 20: Lemma 6.12. The dart d_4 enters $d_3 \circ d_2$ from the right. (a) In this case, d_4 is the dart c that precedes A_1 in A. (b) In this case, d_4 is the dart of A_1 whose head is $tail(d_2)$.

enclosed by $A_1 \circ rev(P)$ and therefore by $A_1 \circ P'$, we infer that $tail(d)$ belongs to $A_1 \circ P'$, proving the claim.

Every dart of W_1 is enclosed by $A_1 \circ rev(P)$, so $end(W_1)$ belongs to $A_1 \circ rev(P)$. If $end(W_1)$ were not on A_1, then $end(W_1)$ would be an internal vertex of P and an internal vertex of P', again contradicting Property 6.7. Thus W_1 ends on A_1, as shown in Figure 19(b).

Since A is a simple path ending at t, A_1 is a subpath of A,and b occurs on A after A_1, it follows that A_1 does not include t. Therefore W_1 is a proper prefix of F_t. Let d_1 be the dart of F_t immediately after W_1.

Let $D = rev(W_1 \circ d_1) \circ R_a[\cdot, head(b)]$. We claim that some dart of A_1 enters D from the right. This argument is illustrated in Figure 20. Note that D contains b, which also belongs to A_1. Let d_2 be the first dart in $D[tail(d_1), \cdot]$ that is not in $rev(A_1)$, and let d_3 be its predecessor dart in D.

The dart d_3 either is the reverse of a dart of A_1 or is not enclosed by $A_1 \circ rev(P)$. The dart d_2 is enclosed by $A_1 \circ rev(P)$ and is not the reverse of a dart of A_1. If $tail(d_2) = start(A_1)$, then let $d_4 = c$; then d_4 is strictly enclosed in Δ_a, and $head(c) = tail(d_2)$. Otherwise, let d_4 be the dart of A_1 whose head is $tail(d_2)$. In either case, d_4 enters $d_3 \circ d_2$ from the right, as illustrated in Figure 20. This is Situation 3 of Lemma 6.1, contradicting that lemma.

Case 2: $A_1 = A[\cdot, head(b)]$. This case is illustrated in Figure 21. Let F_s be any s-to-$end(R_a)$ flow path. Let A_2 be the maximal suffix of A_1 that does not cross F_s. Let $F'_s = F_s[start(A_2), \cdot]$. Since $start(A_2)$ is not strictly enclosed by Δ_a, F'_s starts outside the interior of Δ_a, and so by Property 6.7 no part of F'_s is interior to Δ_a. Let $C = F'_s \circ rev(R_a[tail(b), \cdot]) \circ rev(A_2[\cdot, tail(b)])$. By the choice of A_2, C is non-self-crossing. By Part 2 of Lemma 6.1, $rev(C)$ is c.c.w.

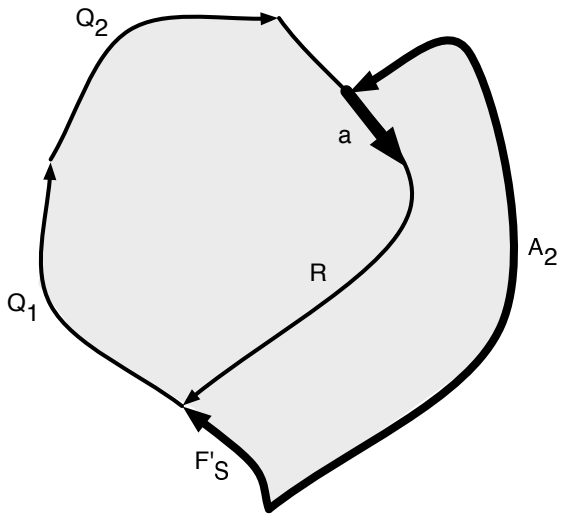


Figure 21: Lemma 6.12, Case 2.

and so C is c.w. By applying the Composition Lemma (Lemma 4.3) to Δ_a and $rev(A_2[\cdot, tail(b)]) \circ F'_s$, we get a non-self-crossing cycle C_1. Let $Q^1 = F'_s \circ Q^1_a$ and $R' = R_a[\cdot, tail(b)] \circ rev(A_2)$. Then $C_1 = Q^1 \circ Q^2_a \circ R'$ is an obstruction for a after augmentation since $rev(A_2)$ is residual after augmentation. This proves the lemma. □

This completes the proof of the Unusability Theorem.

7 Closing Remarks

In closing, we mention two more general versions of max-flow. The first is maximum flow subject to vertex capacities. The best known result is by Khuller and Naor [25]. The second problem is maximum flow with multiple sources and/or sinks. As Miller and Naor point out [29], planarity is not preserved by the traditional reduction from multiple-source/sink max flow to single-source/sink max-flow. Is there a planarity-exploiting algorithm for either of these problems?

References

[1] U. Acar, G. Blelloch, R. Harper, J. Vittes, and S. Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 531–540, 2004.

[2] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005.

[3] T. Biedl, B. Brejová, and T. Vinař. Simplifying flow networks. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 192–201, 2000.

[4] G. Borradaile and P. Klein. An $O(n \log n)$-time algorithm for maximum st-flow in a directed planar graph. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 524–533, 2006.

[5] E. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[6] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7:646, 1960.

[7] J. Edmonds and R. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[8] P. Elias, A. Feinstein, and C. Shannon. A note on the maximum flow through a network. *IEEE Transactions on Information Theory*, 2(4):117–119, 1956.

[9] D. Eppstein, G. Italiano, R. Tamassia, R. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1 – 11, 1990.

[10] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, near linear time. In *Proceedings of the 42th Annual Symposium on Foundations of Computer Science*, pages 232–241, 2001.

[11] C. Ford and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[12] G. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[13] A. Goldberg. Recent developments in maximum flow algorithms. In *Proceedings of the 6th Scandinavian Workshop on Algorithm Theory*, pages 1–10, 1998.

[14] A. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998.

[15] A. Goldberg and R. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[16] T. Harris and F. Ross. Fundamentals of a method for evaluating rail net capacities. Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California, 1955.

[17] R. Hassin. Maximum flow in (s, t) planar networks. *Information Processing Letters*, 13:107, 1981.

[18] R. Hassin and D. B. Johnson. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM Journal on Computing*, 14:612–624, 1985.

[19] L. Heffter. Über das problem der nachbargebiete. *Mathematische Annalen*, 38:477–508, 1891.

[20] M. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[21] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[22] A. Itai and Y. Shiloach. Maximum flow in planar networks. *SIAM Journal on Computing*, 8:135–150, 1979.

[23] D. B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.

[24] 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 *Proceedings of the 20th Annual Allerton Conference on Communication, Control, and Computing*, pages 898–905, 1982.

[25] S. Khuller and J. Naor. Flow in planar graphs with vertex capacities. *Algorithmica*, 11(3):200–225, 1994.

[26] S. Khuller, J. Naor, and P. Klein. The lattice structure of flow in planar graphs. *SIAM Journal on Discrete Mathematics*, 6(3):477–490, 1993.

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

[28] A. Kotzig. *Súvislosť a Pravidelná Súvislosť Konečných Grafov*. PhD thesis, Vysoká Škola Ekonomická, Bratislava, 1956.

[29] G. L. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24(5):1002–1017, 1995.

[30] J. Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM Journal on Computing*, 12:71–81, 1983.

[31] H. Ripphausen-Lipa, D. Wagner, and K. Weihe. Efficient algorithms for disjoint paths in planar graphs. In W. Cook, L. Lovasz, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 295–354. AMS, 1995.

[32] A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[33] D. Sleator and R. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[34] D. Sommerville. *An introduction to the geometry of n dimensions*. London, 1929.

[35] R. Tarjan and R. Werneck. Self-adjusting top trees. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 813–822, 2005.

[36] K. Weihe. Maximum (s, t)-flows in planar networks in $O(|V|log|V|)$ time. *Journal of Computer and System Sciences*, 55(3):454–476, 1997.

[37] H. Whitney. Planar graphs. *Fundamenta mathematicae*, 21:73–84, 1933.

[38] J. Youngs. Minimal imbeddings and the genus of a graph. *Journal of Mathematical Mechanic*, 12:303–315, 1963.