AN ABSTRACT OF THE THESIS OF

Anna Harutyunyan for the degree of Master of Science in Computer Science presented on November 30, 2012.

Title: Maximum Flow in Planar Digraphs

Abstract approved: ______________________________

Glencora Borradaile

Worst-case analysis is often meaningless in practice. Some problems never reach the anticipated worst-case complexity. Other solutions get bogged down with impractical constants during implementation, despite having favorable asymptotic running times. In this thesis, we investigate these contrasts in the context of finding maximum flows in planar digraphs. We suggest analytic techniques that adapt to the problem instance, and present a structural property that concludes equivalence between shortest paths and maximum st-flow in planar graphs.

The best known algorithm for maximum st-flow in directed planar graphs is an augmenting-paths algorithm with $O(n)$ iterations. Using dynamic trees, each iteration can be implemented in $O(\log n)$ time. Long before, Itai and Shiloach showed that when s and t are on the boundary of a common face, the $O(n)$-iteration augmenting-paths algorithm is equivalent to Dijkstra's algorithm in the graph's dual: the max st-planar st-flow problem can be solved with one single-source shortest-path computation. In this thesis we show that (a) when s and t are separated by p faces, the max st-flow can be found with at most $2p$ single-source shortest-path computations, which, using the linear-time shortest-paths algorithm for planar graphs, results in an $O(np)$-time algorithm, and (b) that the equivalence between augmenting-paths and Dijkstra's extends to the most general non-st-planar digraphs, using their half-infinite universal cover graph.

Maximum Flow in Planar Digraphs

by

Anna Harutyunyan

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 30, 2012
Commencement June 2013

ACKNOWLEDGEMENTS

I am infinitely grateful to Cora Borradaile who taught me how to do research, for her contagious enthusiasm, for the hours spent at the whiteboard together, and for being a role model. I look forward to working with her in the future.

I would like to thank my committee members and former professors for their guidance, assistance, and encouragement.

I am grateful to the department staff for making all formalities seem easy.

I also extend sincere thanks to friends and fellow graduate students who (sometimes involuntarily) engaged in discussions and provided invaluable insights: Theresa, Boris, Christophe, Mohamed, Behrouz, William, Ben and others. Lastly, I thank my family for their enduring support.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

To my family,
and people I took coffee breaks with.

Chapter 1: Introduction

Recently, James Orlin announced an $O(nm)$-time[1] algorithm for finding maximum flow in general networks, solving a problem open for over 50 years. Maximum flow is motivated by a number of practical problems, with optimization of traffic, transportation and communication networks being the classic ones. It is used in solving various algorithmic problems, such as finding the maximum number of edge- or vertex-disjoint paths, minimum spanning trees, bipartite matchings and others [2], where max flow algorithms are used as a black-box.

In general graphs, the first strongly polynomial-time algorithm was developed in 1972 by Edmonds and Karp [10]. This was followed by faster and faster algorithms with improvements made nearly every few years, until King, Rao and Tarjan developed their $O(nm + n^{2+\epsilon})$ and $O(nm \log_{m/(n \log n)} n)$ algorithms in 1992 [29] and 1994 [30], respectively. These bounds match $O(nm)$ except for sparse graphs. No better strongly polynomial algorithm for sparse graphs has been developed for 18 years, until Orlin's recent result. Orlin not only gets the $O(nm)$ bound, but also an $O(n^2/\log n)$ bound for sparse graphs ($m = O(n)$).

Planar graphs are very sparse. Sparseness and a number of planarity-induced properties allow for faster algorithms and more efficient computation on the plane. These results often generalize to more general graph families, such as graphs embeddable on bounded-genus surfaces, or graphs with forbidden minors. Turns out, the real world is often flat; planar (and nearly planar) graphs serve as a reasonably accurate model for a number of applications, including road or cable networks, images and VLSI circuits.

Maximum flow in planar graphs can be found even quicker than in sparse graphs. And in fact, the very origin of the maximum flow problem lies in planar graphs. In the mid-1950s Air Force researchers Harris and Ross wrote a classified report showing the rail network that connected the Soviet Union with its satellite countries of the Eastern Block. Regions were represented by nodes with their connecting rail segments being the edges. Each edge was assigned a weight that represented its "capacity": the rate at which it could transport materials between its endpoints. Experimentally, they determined the maximum amount of goods that could be transported from Russia into Europe, and the cheapest way to separate the network by removing links (or, in simpler terms, blowing up the rails), which they called the "bottleneck" (Fig. 1.1). This information, being first declassified in 1999 [34], is the first formulation of the Maximum Flow, Minimum Cut problem.

In 1956, Ford and Fulkerson published their seminal work proving the Max Flow, Min Cut Theorem: the value of maximum flow is equal to the value of the minimum cut. Their

[1] n commonly denotes the number of vertices, and m - the number of edges in the input graph.

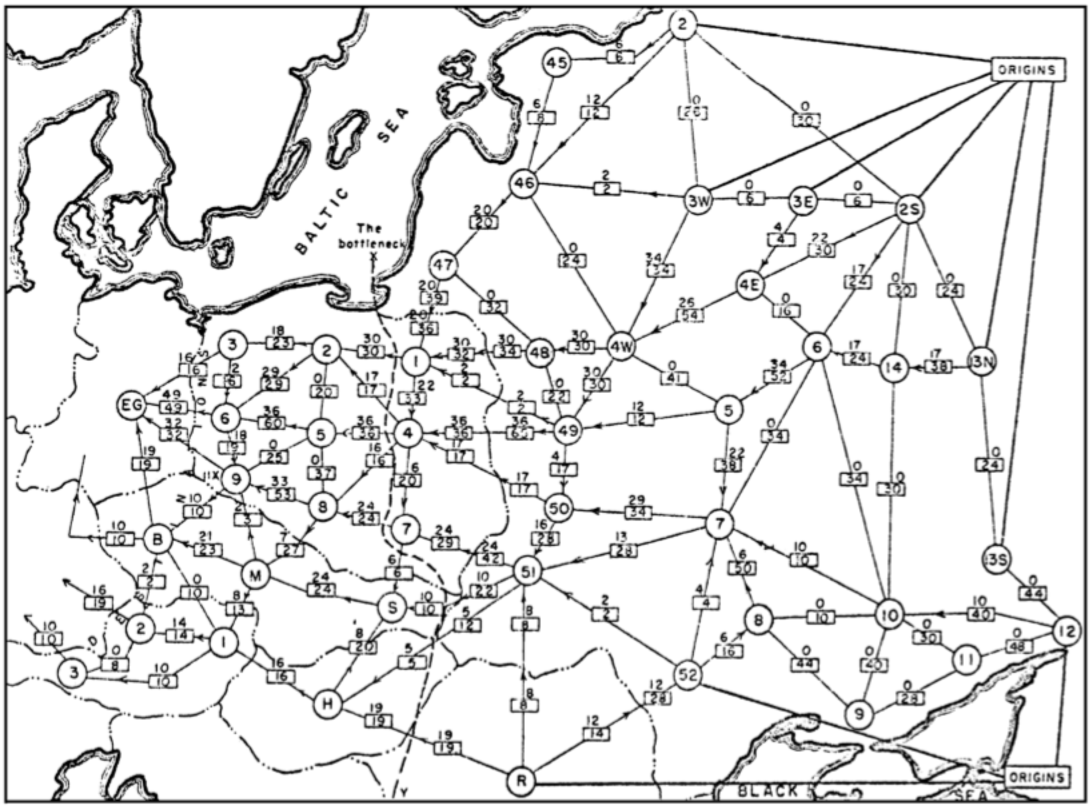

Figure 1.1: Harris and Ross map of the Warsaw Pact rail network. The model graph has 44 vertices and 105 directed edges, with the dotted line representing the "bottleneck".

work also includes an analysis of an augmenting-paths algorithm for st-planar graphs (graphs in which s and t are on the same face) [13]. They show that viewing the graph as being embedded with s as the bottom vertex and t as the top vertex, repeatedly augmenting the leftmost[2] residual s-to-t path will find the max flow in $O(n)$ iterations. Later, Itai and Shiloach showed that this algorithm can be implemented by running Dijkstra's algorithm in the dual graph [21]. Borradaile and Klein showed that a generalization of leftmost to planar graphs in which s and t are not on the same face admits an $O(n)$-iteration augmenting paths algorithm; each iteration can be implemented in $O(\log n)$ time for an $O(n \log n)$-time algorithm [5].

The above results also, of course, apply to undirected planar graphs. Recently, faster algorithms have been developed for undirected graphs. Kaplan and Nussbaum show that, when s and t are separated by p faces, the minimum cut (but not the maximum flow) can be computed in $O(n \log p)$ time [25], and Italiano, Nussbaum, Sankowski and Wulff-Nilsen show that the maximum flow can be computed in $O(n \log \log n)$ time [22]. The history of maximum flow in planar graphs shows that maximum flow and minimum cut have been inseparable in directed graphs but not in undirected graphs: Kaplan and Nussbaum's

[2]Ford and Fulkerson viewed the graph as embedded with s on the left and t on the right and augmented the uppermost path.

recent result for undirected graphs does not yield the flow (Table 1.1).

The Max Flow problem was and still is a fundamental algorithmic challenge, with a rich history and continuously evolving solutions.

Year	**Restriction**	**Year**	**Reference**
1956	st-planar	$O(n^2)$	Ford and Fulkerson [13]
1979	st-planar	$O(n \log n)$	Itai and Shiloach [21]
1982	flow of given value	$O(n\sqrt{n} \log n)$	Johnson and Venkatesan [23]
1983	value, undirected	$O(n \log^2 n)$	Reif [33]
1985	undirected	$O(n \log^2 n)$	Hassin and Johnson [18]
1987	st-planar	$O(n\sqrt{\log n})$	Hassin [17] using Frederickson [14]
1997	st-planar	$O(n)$	Hassin [17] using Henzinger et al. [19]
1997	undirected	$O(n \log n)$	Hassin and Johnson [18] using Henzinger et al. [19]
2001		$O(n \log^3 n \log C)$	Miller and Naor [32] using Fakcharoenphol and Rao [12]
2006		$O(n \log n)$	Borradaile and Klein [4]
2011	undirected	$O(n \log \log n)$	Italiano et al. [22]
2011	undirected, minimum cut	$O(n \log p)$	Kaplan and Nussbaum [25]

Table 1.1: History of Planar Maximum Flow and Minimum Cut Algorithms

Organization and Contributions

In Chapter 2, we give the background and preliminaries necessary to understand this thesis.

In Chapter 3, we generalize the leftmost augmenting-path algorithm for max st-flow in directed planar graphs. The algorithm finds the leftmost s-to-t residual path P, but, instead of augmenting just that path, it augments all the s-to-t paths that do not cross P. This is accomplished with a single shortest-path computation. We show that the number of times these paths cross the band of p faces separating s from t is at most p in each direction; the algorithm therefore takes $O(p)$ shortest-path computations. Such an algorithm is called adaptive since the running time measures to not just the input size, but adapts to another parameter that captures the difficulty of the input instance.

In Chapter 4, we extend the equivalence between flow and shortest-paths to the most general case of st-flow in planar graphs: graphs in which s and t are not on the same face. We do this by augmenting paths in the half-infinite universal cover of the graph. Contrary to the standard approach, we do not explicitly update the residual capacities in the graph, instead moving forward in its cover. The correspondence concludes that any planar maximum st-flow is witnessed by dual shortest paths.

Chapter 2: Preliminaries

We give the basic definitions and concepts necessary to understand this thesis. For more detailed background on graphs, flow, and planarity please refer to [2] and [7]. We extend any function or property on elements to sets of elements in the natural way.

2.1 Background

2.1.1 Graphs

An *undirected graph* is an ordered pair $G = (V, E)$, where V is a set of *vertices*, and E is a set of *edges*, where an edge is a pair of vertices. For two vertices u and v, we denote the edge between them by uv. Despite finding flows in directed graphs throughout this thesis, we refer to the underlying undirected graph $G = (V, E)$. Each edge in E has two corresponding oppositely-directed *darts*. We define $rev(\cdot)$ to be a function that takes each dart to the corresponding dart in the opposite direction. The *head* and *tail* of a dart d in G are vertices such that d is oriented from tail to head.

Walks are directed: they are ordered sets of darts with the head of a dart in the walk being the tail of the next dart in the walk. We say that a vertex u comes *before* a vertex v on a walk P, if the dart whose head is u comes before the dart whose head is v in P; the tail of the first dart of P is before all vertices on P. *After* is defined analogously. If P and Q are two walks with the last vertex of P being the first vertex of Q, we denote the concatenation of these walks by $P \circ Q$. If P is a walk and u, v are a pair of vertices on it, $P[u, v]$ denotes the subset of P between the dart whose tail is u and the dart whose head is v. $P[\cdot, v]/P[u, \cdot]$ corresponds to $P[u, v]$, where u/v is the first/last vertex on P. A *path* is a walk in which each dart is used at most once. A path is *simple* if each vertex is the head of at most one dart in the path. A *cycle* is a path whose start and endpoints are the same.

2.1.2 Flow

Let c be a capacity function on the darts of G. Capacities are directed: $c(d)$ need not be equal to $c(rev(d))$. A *flow assignment* $\mathbf{f}(\cdot)$ is a function on the darts of G. The flow assignment $\mathbf{f}(\cdot)$ *respects the capacity* of dart d, if $\mathbf{f}(d) \leq c(d)$. A flow assignment that respects the capacities of all darts is a *pseudoflow*. For a given flow assignment $\mathbf{f}(\cdot)$, the *net inflow* (or just *inflow*) of a vertex v is the sum of the flow of darts whose head is v less the sum of the flow on darts whose tail is v. *Net outflow* is defined analogously. Vertices with positive inflow are *excess vertices*, and vertices with negative inflow are *deficit vertices*. The flow assignment $\mathbf{f}(\cdot)$ is *conserved* at a vertex v, if the net inflow is zero. A pseudoflow that only contains vertices with non-negative inflow is a *preflow*[1]. A pseudoflow that only contains vertices with non-negative outflow is a *postflow.* A pseudoflow that is conserved at all vertices is a *feasible circulation.* A *feasible flow* is a circulation that need not be conserved at the source and sink vertices. The *value* of a flow is the sum of the flow on darts whose tail is a source less the sum of the flow on darts whose head is source. A *maximum flow* is a feasible flow of maximum value. A *maximum preflow* is a preflow that maximizes the flow into the sinks. Similarly, a *maximum postflow* is a postflow that maximizes the flow from the sources.

Given capacities c and a flow $\mathbf{f}$, we define the *residual capacities of c with respect to* $\mathbf{f}$ as $c'(d) = c(d) - \mathbf{f}(d) + \mathbf{f}(rev(d))$. We shorten this to *residual capacities* when c and $\mathbf{f}$ are clear from context. In Chapter 3 we will be considering a sequence of capacities $c_0, c_1, \cdots$ where c_i are the residual capacites of c_{i-1} with respect to some flow. We refer to the flow that takes c_{i-1} to c_i. A dart is *residual with respect to capacities c* if $c(d) > 0$. We shorten this to *residual* when the capacities are clear from context.

We will use the following two lemmas that hold for general graphs. The forward direction follows from the definition of maximum preflow (postflow). The reverse direction follows from the Max Flow, Min Cut Theorem: if there are no residual paths from sources or vertices with excess flow to sinks or vertices with deficit flow, then there is a saturated cut separating the sources from the sinks and the preflow (postflow) cannot be increased [16].

Lemma 2.1.1. *A preflow is maximum if and only if there is no residual path from a source to a sink or from an excess vertex to a sink.*

Lemma 2.1.2. *A postflow is maximum if and only if there is no residual path from a source to a sink or from a source to a deficit vertex.*

2.1.3 Planar graphs

A planar graph is a graph for which there exists a planar embedding. A planar embedding of a graph is the 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 set of contiguous points in the plane/on the sphere that are not in the image of the vertices or arcs is a *face.* For an embedding on the plane, there is one *infinite face.* For an embedding on the sphere, an arbitrary face can be designated as the infinite face. We denote the infinite face by f_∞.

Aside from this topological definition, one can also define embeddings combinatorially [9]. A combinatorial embedding, or a *rotation system*, is given by a permutation π, such that for every 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 this formulation is crucial in implementing planar graph algorithms, it will not be used

[1]Preflows were first introduced by Karzanov in 1974 [27], and popularized by Goldberg and Tarjan in 1988 [16].

explicitly in this thesis.

Planar graphs can also be characterized by the minors they do not contain: the complete graph with 5 vertices, K_5, and the complete bipartite graph with 6 vertices, $K_{3,3}$ [31, 37].

Duality. The *dual* graph of a planar graph G, is another planar graph G^* whose vertices are the faces of G and vice versa. Two vertices in G^* are connected by an edge if and only if their corresponding faces share an edge in G. Thus there is a one-to-one correspondence between edges of G and edges of G^*. See Fig. 2.1. We use the following classic result on planar graphs, illustrated in Fig. 2.2[2].

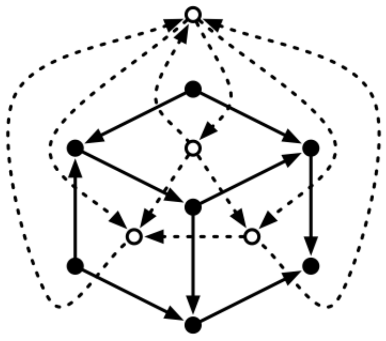

Figure 2.1: A planar graph and its dual: the primal graph is given by solid vertices and solid arcs and the dual is given by open vertices and dotted arcs.

Theorem 2.1.3 (Cycle-Cut Duality [38]). *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.*

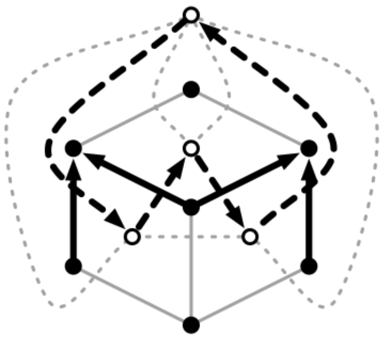

Figure 2.2: The primal is given by solid edges and the dual by dotted edges. The dark bold (directed) darts form a simple directed cycle in the dual and a directed cut in the primal.

[2]Both Figures 2.1 and 2.2 first appear in [3].

Entering and leaving. Suppose a, b, and d are darts such that $\mathit{head}(a) = \mathit{tail}(b) = \mathit{head}(d) = v$: we say d *enters* $a \circ b$ at $\mathit{head}(a)$. If the clockwise ordering of these darts around v is a, b, d, then d enters $a \circ b$ from the right and $\mathit{rev}(d)$ leaves $a \circ b$ from the right. Likewise, if the clockwise ordering of these darts around v is a, d, b, then d enters $a \circ b$ from the left and $\mathit{rev}(d)$ leaves $a \circ b$ from the left.

Crossing. We say that P crosses Q at X if X is a maximal subpath of Q such that either X or $\mathit{rev}(X)$ is a subpath of P and:

- P enters Q from the right at $\mathit{start}(X)$ and P leaves Q from the left at $\mathit{end}(X)$ in which case P *crosses* Q *from right to left*, or
- P enters Q from the left at $\mathit{start}(X)$ and P leaves Q from the right at $\mathit{end}(X)$ in which case P *crosses* Q *from left to right.*

By definition, X cannot be a prefix or suffix of either P or Q. A crossing is *simple* if X is a single vertex. 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 its subpaths, P does not cross Q. Note that, for any face f, the boundary of f is a non-self-crossing cycle. The notions of entering and crossing are illustrated in the Figure 2.3.

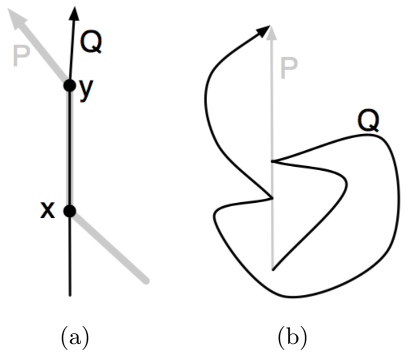

Figure 2.3: (a) P crosses Q from right to left: P enters Q on the right at x, and P leaves Q on the left at y. (b) P and Q are non-crossing

Cutting. For a path P, $\mathit{left}(P)$ is the set of edges whose darts enter or leave P from the left ($\mathit{right}(P)$ is defined analogously). $G \not\!\!\!\times P$ is the graph $G = (V, E \setminus \mathit{left}(P))$. All the vertices of P are on a common face of $G \not\!\!\!\times P$.

Clockwise and leftmost. We call ϕ a *potential assignment*, and $\phi[f]$ - the *potential* of face f. Corresponding to every circulation in a planar graph, there is a potential assignment such that the flow on a dart d is given by the difference between the face on the left side and right side of d, and the potential of the infinite face is 0. A circulation is *clockwise* if all the potentials are negative. A cycle is *clockwise* if the circulation that pushes one unit of flow on each dart of the cycle is clockwise. The circulation (the flow) such that every clockwise cycle is non-residual is *leftmost*. Khuller, Naor and Klein showed that circulations in planar graphs form a finite distributive lattice, with the leftmost circulation being its unique minimum [28].

A path (a walk) A is *left of* a path (a walk) B if $A \circ rev(B)$ is a clockwise circulation. A path (a walk) is *leftmost* if there are no paths (walks) left of it. (Counterclockwise, right, and rightmost are defined symmetrically.)

We use the following results implicitly throughout.

Lemma 2.1.4 (Lemma 4.4 [5])**.** *Let G be a graph with no clockwise cycles. If P is a leftmost walk, then P is a simple path.*

Lemma 2.1.5 (Lemma 4.5 [5])**.** *Every subpath of a leftmost path is a leftmost path.*

Lemma 2.1.6 (Corollary 3.7 [5])**.** *Leftmost flow is acyclic.*

2.2 Converting a pseudoflow into a feasible maximum flow

We start by stating a property that will be used in the analysis of this section:

Lemma 2.2.1 (Lemma A.3 [6])**.** *Let $\mathbf{f}$ be a pseudoflow in G with sink set X. Let A, B be two disjoint sets of nodes. If there are no residual paths in G from A to $B \cup X$, then there are no residual paths in G from A to B w.r.t. to $\mathbf{f}$.*

We refer to the above lemma for node sets W, Y, Z as *sinks lemma*(W,Y,Z).

Let $\mathbf{f}$ be a pseudoflow in a planar graph G with node set V, sources S and sinks T. Let V^+ denote the set of excess nodes, and let V^- denote the set of deficit nodes. Suppose there are no $S \cup V^+$-to-$T \cup V^-$ residual paths. In this section, we show how to convert $\mathbf{f}$ into a maximum feasible flow $\mathbf{f}'$. This procedure was first described for planar graphs by Johnson and Venkatesan [23], and then by Borradaile et al. [6]. The original description of the procedure took $O(n \log n)$ time, but with the use of the flow-cycle canceling technique[3] of Kaplan and Nussbaum [26], the running time is $O(n)$.

Since $\mathbf{f}$ is acyclic, there exists a topological ordering on the nodes of the graph G. Let v be the last member of V^+ in the topological ordering, and let d be an arbitrary dart that carries flow into v. We reduce the flow on d by the amount of excess on v. If v has more excess than $c(d)$, we set $\mathbf{f}(d) = 0$. $\mathbf{f}(rev(d))$ is handled accordingly. The flow assignment $\mathbf{f}$ maintains the invariant of no $S \cup V^+$-to-$T \cup V^-$ residual paths by *sinks lemma*$(S \cup V^+, T \cup V^-, \{v\})$.

As long as v is in V^+, there must be a dart d which carries flow to v. By changing the flow on d we cannot add to V^+ a new node that appears later than v in the topological ordering. We repeat this process until V^+ is empty. Since we reduce the flow on each dart at most once, this takes linear time. Next we handle V^- in a symmetric way by repeatedly fixing the first vertex of V^- in the topological ordering.

[3]We note, however, that since we are concerned with leftmost flows, and leftmost flows are acyclic, this initial step will not be needed.

The total running time is $O(n)$ and, since V^+ and V^- are both empty, we get from the invariant of no residual $S \cup V^+$-to-$T \cup V^-$ paths that the resulting pseudoflow is a feasible flow. This is the required maximum flow $\mathbf{f}'$.

2.3 Hassin's algorithm for maximum st-planar flow

A graph G is st-planar if the source and the sink are on the boundary of the common face. In this section, we give a brief description of Rafael Hassin's algorithm for finding maximum st-planar st-flow [17], since we refer to it liberally throughout this thesis.

Hassin's algorithm starts by transforming the max flow problem into a max saturating circulation problem: it embeds an extra infinite-capacity directed 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. In the dual graph G^*, compute a shortest-path tree rooted at t^*, where the length of a dual dart is defined as the capacity of the primal dart. Let $\phi[\cdot]$ denote the shortest path distances from t^* in G^*. Consider the flow:

$$\mathbf{f}[d'] = \phi[head_{G^*}(d')] - \phi[tail_{G^*}(d')], \text{ for all darts } d'.$$

After removing the artificial arc a from G, $\mathbf{f}$ is a maximum feasible flow from s to t in G, and ϕ is the potential assignment that induces $\mathbf{f}$.

2.4 Universal cover

We define the *universal covering graph* (or, simply, the *universal cover*) of G. This is an instance of the more general universal covering map concept from topology [36], but our use of it can be viewed independently via the definitions presented below.

The construction of our universal cover is similar to the one described in Section 2.4 of Erickson's analysis of the leftmost-path algorithm of Borradaile and Klein [11]. Erikson's cover was of the dual graph, whereas we remain in the primal.

Given an embedding of $G = (V, E)$ on the sphere, consider the cylinder obtained by deleting an arbitrary face adjacent to s and an arbitrary face adjacent to t in G. We construct the universal cover with respect to a simple s-to-t path P. We will later show that the universal cover does not depend on which s-to-t path we use. Intuitively, the universal cover is obtained by cutting the cylinder along P into a rectangle, and pasting copies of these rectangles along P, creating a doubly-infinite strip.

Formally, the universal cover is an infinite graph $\mathcal{G}_P = (\mathcal{V}_P, \mathcal{E}_P)$ corresponding to a doubly-infinite sequence of copies of G: $\dots, G_P^{-1}, G_P^0, G_P^1, \dots$ (or, simply, $\dots, G_{-1}, G_0, G_1, \dots$, if P is clear from context), where $G_P^{i-1} \cap G_P^i = P^i$ for all i. We define this more precisely using the function $\lambda(uv, P)$ that indicates when vertex $u \in P$ and dart $uv \in \mathit{left}(P)$.

$$\mathcal{V}_P = \{v_i | v \in V, i \in \mathbb{Z}\}$$
$$\mathcal{E}_P = \{u_{i+\lambda(uv,P)} v_{i+\lambda(vu,P)} | uv \in E\}$$

A set of edges $X \in E$ maps to an infinite family $\mathcal{X}_P = \{\dots, X_P^{-1}, X_P^0, X_P^1, \dots\}$ (or, simply, $\ldots, X_{-1}, X_0, X_1, \ldots$, if P is clear from context) in $\mathcal{G}_P$.

Lemma 2.4.1. *Let Q be a simple path starting and ending on P and sharing no darts with P. Let a and b be the first and last, resp., darts of Q. Consider one copy Q' of Q in $\mathcal{Q}_P$.*

1. *If $a, b \in$ left(P), Q' starts and ends on P^j for some j.*
2. *If $a, b \in$ right(P), Q' starts and ends on P^j for some j.*
3. *If $a \in$ left(P), $b \in$ right(P), Q' starts on P^{j+1} and ends on P^j for some j.*
4. *If $a \in$ right(P), $b \in$ left(P), Q' starts on P^j and ends on P^{j+1} for some j.*

Proof. Let a' and b' be the first and last darts of Q'. Let $tail(a') = u_i$ and $head(b') = v_j$. The first case of the lemma is equivalent to showing that $i = j$. Since $a, b \in left(P)$ and $u \in P$, $v \in P$, $\lambda(a, P) = \lambda(rev(b), P) = 1$. The second case is similar. The third case is equivalent to showing that $i = j + 1$. Since $a \in left(P)$, $b \in right(P)$ and $u \in P$, $v \in P$, $\lambda(a, P) = 1$, $\lambda(rev(b), P) = 0$. The last case is symmetric. $\square$

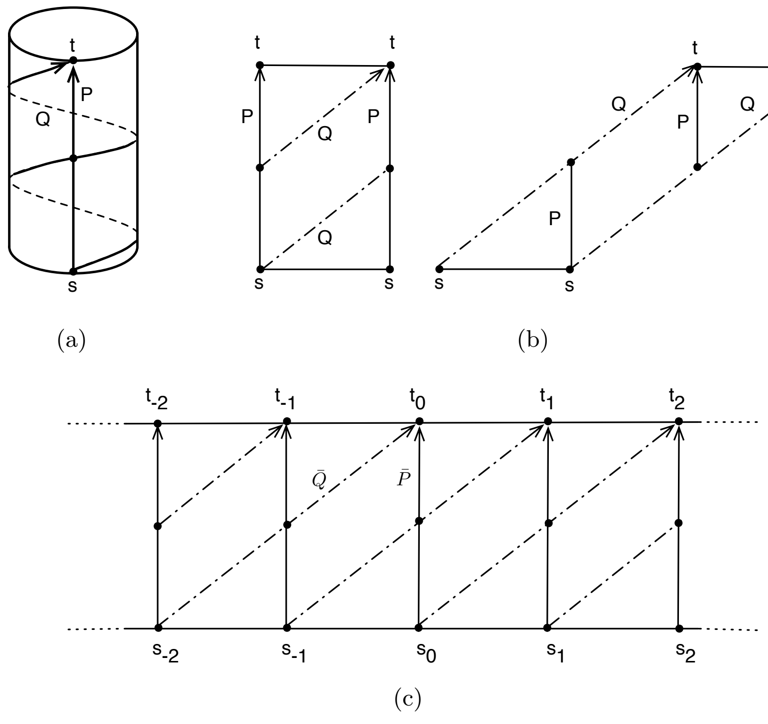

Figure 2.4: (a) G embedded on a cylinder with s-to-t paths P and Q. (b) Embeddings of G that correspond to "cutting" along P and Q. (c) The universal cover graph of G: $\mathcal{G}_{\mathcal{P}} = \mathcal{G}_{\mathcal{Q}}$, with the labeling on t inherited from $\mathcal{G}_{\mathcal{P}}$.

Note that due to the infinite structure of $\mathcal{G}_{\mathcal{P}}$, the universal cover is independent of the choice of P, up to labeling of vertices. In fact, their embeddings are the same by inheritance from G. See Fig. 2.4. Formally:

Lemma 2.4.2. *Let P and Q be arbitrary, simple s-to-t paths in G. $\mathcal{G}_P$ is isomorphic to $\mathcal{G}_Q$.*

Proof. Consider the function $f(u_j v_k) = u_{j-\lambda(uv,P)+\lambda(uv,Q)} v_{k-\lambda(vu,P)+\lambda(vu,Q)}$. This is a bijection between edge sets $\mathcal{E}_P$ and $\mathcal{E}_Q$ since edges in $\mathcal{E}_P$ take the form $u_{i+\lambda(uv,P)} v_{i+\lambda(vu,P)}$. Therefore $\mathcal{G}_P$ is isomorphic to $\mathcal{G}_Q$. □

Let Q be an s-to-t path in G. There is a corresponding family of s^i_P-to-t^{i+j}_P paths in $\mathcal{G}_P$ for all i and some j that we define as the *lift* of Q. We refer to the unique particular path in this family for which $i + j = 0$ as $\overline{Q}_P$.

The above isomorphism implies:

Corollary 2.4.3. *Consider universal covers $\mathcal{G}_P$ and $\mathcal{G}_Q$. Let A and B be s-to-t paths in G. If $\overline{A}_P$ starts at s_j and $\overline{B}_P$ starts at $s_{j+\ell}$, then if $\overline{A}_Q$ starts at $s_{j'}$, $\overline{B}_Q$ must start at $s_{j'+\ell}$.*

For a simple path Q, we can bound the index at which $\overline{Q}_P$ starts:

Lemma 2.4.4. *Let P and Q be arbitrary simple s-to-t paths in G. $\overline{Q}_P$ in $\mathcal{G}_P$ must start at s_j, s.t. $|j| \leq |P|$.*

Proof. For simplicity, assume j to be non-negative; the argument for when j is negative is symmetric. By construction of $\mathcal{G}_P$, an s_j-to-t_0 path must cross each $P^{j-1}, \ldots, P^1$ at least once in $\mathcal{G}_P$, and therefore Q must cross P at least $j-1$ times in G. By the pigeonhole principle, if $j - 1 > |P| - 1$, Q must cross P at some vertex u twice, creating a cycle. Contradiction. □

2.5 Leftmost-path algorithm

```
MaxLeftmostFlow
    Designate a face adjacent to $t$ as $f_\infty$.
    Saturate the clockwise cycles.(LeftmostCirculation [5])
    While there is a residual $s$-to-$t$ path,
        saturate the leftmost such path.(MaxFlow [5])
```

Table 2.1: MaxLeftmostFlow Abstract Algorithm

The leftmost-path algorithm of Borradaile and Klein is a direct generalization of the uppermost-path algorithm for non-st-planar graphs [5]. It takes a leftmost flow of zero value, and by way of repeatedly augmenting s-to-t residual paths obtains a leftmost maximum flow. At an abstract level the algorithm is given in Table 2.1.

MaxLeftmostFlow runs in $O(n \log n)$ time. The crux of the analysis, the Unusability Theorem, states that each edge may only get saturated once in each direction, implying a linear bound on the number of augmentations. Each augmentation can be implemented in $O(\log n)$ time with the dynamic tree data structure, giving the stated running time.

Chapter 3: Adaptive Analysis

Worst-case analysis is often overly pessimistic. A classic example is the familiar Quicksort algorithm: a sorting algorithm with worst-case running time of $O(n^2)$ and average-case running time of $O(n \log n)$, that is the most efficient in practice [20]. Average-case analysis, while being much more accurate, is often difficult and cumbersome to perform. Adaptive analysis is an alternative.

Analysis is called *adaptive* when it "adapts" to some parameter that describes the inherent difficulty or easiness of an instance. This is a natural generalization from *output-sensitive* analysis, in which the running time is expressed in terms of the output. Output-sensitive (and later: adaptive) analysis techniques have seen their iconic use in the context of solving the Convex Hull problem: given a set of X points in the Euclidean plane (or Euclidean space), find the smallest set of points Y whose convex hull contains X. Adaptive techniques have advanced the running time from $O(n \log n)$ to $O(nh)$, $O(n \log h)$, and, eventually, $O(nH(x_1, \ldots, x_h))$, where h is the size of the resulting envelope Y, x_i is a point on it, and $H(x_1, \ldots, x_h)$ denotes entropy. For more background and results on adaptive techniques on problems in computational geometry, refer to [1].

A planar graph is "easy" in the context of finding flow, if s and t are on the same face. Indeed, in this case the problem of finding maximum flow reduces to a single-source shortest-path computation in the dual, which can be done in linear time [19]. What if s and t are p faces apart? The parameter p was first introduced by Itai and Shiloach [21] who gave an $O(np \log n)$ algorithm for finding the flow of a known value. Later, Johnson and Venkatesan [24] gave an $O(np \log n)$ algorithm for finding the flow without knowing its value in advance. This algorithm has two bottlenecks: single-source shortest-paths and removing flow-cycles. The first bottleneck was addressed by Henzinger et al. [19] and the second by Kaplan and Nussbaum [26], reducing the running time of Johnson and Venkatesan's algorithm to $\Theta(np)$. This beats Borradaile and Klein's $O(n \log n)$ algorithm, when $p = o(\log n)$.

Recently, Kaplan and Nussbaum generalized Reif's divide and conquer techniques for finding the minimum cut (or, rather, the shortest separating cycle in the dual) in undirected planar graphs [25]. This led to an $O(n \log p)$-time algorithm for finding the minimum cut in undirected planar graphs. Obtaining the flow in matching running time remains an open problem.

In the remainder of this chapter we present an $O(np)$-time algorithm for finding maximum flow in directed planar networks. While offering only a small improvement compared to the algorithm of Johnson and Venkatesan (with the use of modern results), it uses very different techniques, and we believe that the algorithm and its analysis are of independent interest, and have a potential of leading to a better running time.

3.1 MaxAdaptiveFlow Algorithm

Starting with a directed graph with arc capacities c, we consider the underlying undirected graph G and extend c to the darts of G. We take the embedding to have t on the external face.

Our algorithm for max st-flow is presented in Table 3.1.

```
MaxAdaptiveFlow ($G$, $s$, $t$, $c$)
    Let $c_0$ be the residual capacities resulting from saturating
        the clockwise residual cycles of $G$ w.r.t. $c$.
    For $i = 0, 1, \ldots$
        If there is an $s$-to-$t$ residual path in $G$ w.r.t. $c_i$ then,
            let $A_i$ be the leftmost of these paths.
            Let $c_{i+1}$ be the residual capacities resulting from saturating
                the leftmost $st$-flow in $G \text{✂} A_i$ w.r.t. $c_i$.
        Otherwise,
            return the flow defined by $f(d) = \max\{0, c_i(d) - c(d)\}$.
```

Table 3.1: MaxAdaptiveFlow Algorithm

The clockwise saturating circulation required for the first step of MaxAdaptiveFlow and the leftmost st-flows in $G \text{✂} A_i$ can be found with a single-source shortest-path algorithm. See Sections 2.2 and 2.3.1 of Borradaile's dissertation [3] and the work of Khuller, Naor and Klein [28] for details. The leftmost paths can be found by a depth-first left-most search [5].

Let ρ be the number of iterations of MaxAdaptiveFlow. The running time of the algorithm is therefore $O(\rho\,\mathrm{SP}(n))$ where $\mathrm{SP}(n)$ is the time for a single-source shortest-path computation in a planar graph with n vertices; this is bounded by $O(n)$ using the algorithm of Henzinger et al. [19].

The algorithm is correct as it generalizes the augmenting-path algorithm of Ford and Fulkerson [13] and does not complete until there is no residual s-to-t path. We spend the remainder of this chapter bounding ρ in terms of the number of faces separating s from t.

3.2 Analysis

We bound the number of iterations of MaxAdaptiveFlow by showing that the paths $A_0, A_1, \ldots$ first monotonically decrease and then increase in the number of times they cross the shortest (in terms of number of edges) path P_c from s to t.

Notice that A_i must cross A_{i-1} at least once: since A_i is residual in G w.r.t. c_i, A_i cannot be a path in $G \text{✂} A_{i-1}$ (for otherwise it would have been augmented in iteration $i-1$). We show that these crossings can only be from right to left; this will follow from MaxAdaptiveFlow maintaining as an invariant the absence of clockwise residual cycles.

We relate the crossings between A_i and A_{i-1} to A_i and P_c by viewing these paths in the universal cover of G.

3.2.1 Properties of crossing paths

We begin by showing that leftmost residual paths cross other residual paths in a restricted way when there are no clockwise cycles. We denote the sequence of crossings between P and Q by $P \otimes Q$ with ordering inherited from Q. Although the ordering of $P \otimes Q$ and $Q \otimes P$ may not be the same, we have that $|P \otimes Q| = |Q \otimes P|$.

While we only need part 2 of the following theorem, part 1 is used within the proof of part 2 and may be of independent interest.

Theorem 3.2.1 (Leftmost Crossings)**.** *Consider capacities with no residual clockwise cycles. Let P be the leftmost residual s-to-t path, with t on the infinite face, and let Q be an s-to-t path such that $rev(Q)$ is residual. Then:*

1. *The order of crossings is the same along both P and Q. That is, either $X = Y$ or $X = rev(Y)$ where X and Y are the i^{th} crossing in $P \otimes Q$ and $Q \otimes P$, respectively.*

2. *P crosses Q from right to left at X for all $X \in P \otimes Q$.*

Proof. If $|P \otimes Q| = 0$, the theorem is trivially true.

Let $P \otimes Q = \{X_1, X_2, \ldots, X_{|P\otimes Q|}\}$ and define $X_0 = s, X_{|P\otimes Q|+1} = t$. Likewise let $Q \otimes P = \{Y_1, Y_2, \ldots, Y_{|Q\otimes P|}\}$ and define $Y_0 = s, Y_{|Q\otimes P|+1} = t$. For a contradiction to Part 1, let i be the smallest index such that $X_i \notin \{Y_i, rev(Y_i)\}$. Let j be the index such that $Y_j \in \{X_{i+1}, rev(X_{i+1})\}$. Then $j \geq i$ by choice of i.

Let x_i be any vertex in X_i. Since $P[x_{i-1}, x_{i+1}]$ does not cross Q at x_i, $P[x_{i-1}, x_{i+1}]$ does not cross $Q[x_{i-1}, x_{i+1}]$. Since P and $rev(Q)$ are residual, $C_1 = P[x_{i-1}, x_{i+1}] \circ rev(Q[x_{i-1}, x_{i+1}])$ is a simple counterclockwise cycle.

Since there are no crossings in $Q[x_i, x_{i+1}]$, $C_2 = Q[x_i, x_{i+1}] \circ P[x_{i+1}, x_i]$ is a simple cycle. Since P is leftmost residual, $P[x_{i+1}, x_i]$ is left of $rev(Q[x_i, x_{i+1}])$ and C_2 is clockwise.

Since P and Q are simple, C_1 and C_2 do not cross. Therefore it must be the case that either C_1 is enclosed by C_2 or vice versa. See Fig. 3.1.

C_2 is enclosed by C_1 Since P crosses Q at X_{i+1}, $Q[x_{i+1}, \cdot]$ must have a subpath in the strict interior of C_1. Then, a maximal such subpath forms a counterclockwise cycle with a subpath of P, contradicting that P is a leftmost residual path.

C_1 is enclosed by C_2 Since P crosses Q at X_i, $P[x_i, \cdot]$ must enter the strict interior of C_1. Since $P[x_i, \cdot]$ does not cross $Q[x_i, x_{i+1}]$, $P[x_i, \cdot]$ is entirely enclosed by C_1, contradicting that t is on the infinite face.

This proves part 1 of the theorem. Since $Q[x_i, \cdot]$ does not cross $P[x_i, x_{i+1}]$, $Q[x_i, \cdot]$ does not enter the cycle $P[x_i, x_{i+1}] \circ rev(Q[x_i, x_{i+1}])$. Since $P[x_i, x_{i+1}]$ is right of $Q[x_i, x_{i+1}]$, it follows that P enters Q from the right at x_{i+1}. Part 2 follows. □

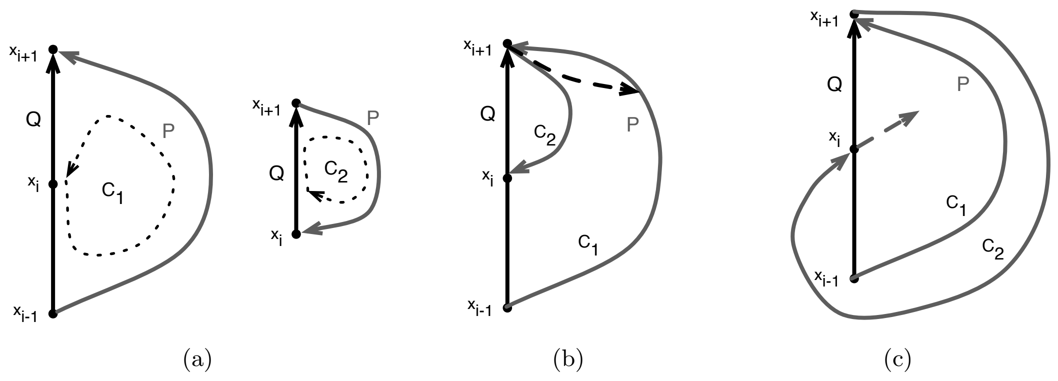

Figure 3.1: (a) Cycles C_1 and C_2 used in the proof of Theorem 3.2.1. (b) Case 1: C_2 is enclosed by C_1. (c) Case 2: C_1 is enclosed by C_2

3.2.2 Left to right progress

Invariant 1. *G has no clockwise residual cycles w.r.t. c_i, for all $0 \le i \le \rho$.*

Proof by induction. If $i = 0$ the invariant holds trivially by definition of c_0. For a contradiction, assume that c_j is clockwise non-residual, but c_{j+1} is not. Let C be a clockwise cycle in G that is residual w.r.t. c_{j+1}. Since G ✂ A_j is clockwise non-residual w.r.t. c_{j+1}, C must use a dart d of *left*(A_j) that enters A_j. By the inductive hypothesis, C is not residual w.r.t. c_j; let a be its last non-residual arc w.r.t. c_j. Let F be an s-to-t path in the flow that takes c_j to c_{j+1} and uses *rev*(a) and let x be the first vertex of F on C after *head*(a). Then: $F[\cdot, x] \circ C[x, head(d)]$ is residual w.r.t. c_j and, since F does not cross A_j, $F[\cdot, x] \circ C[x, head(d)] \circ rev(A_j)[head(d), s]$ is a clockwise cycle, which contradicts A_j being leftmost residual w.r.t. c_j.

□

As a consequence of there being no clockwise residual cycles, we show that the paths $A_0, A_1, A_2, \ldots$ move from left to right.

Lemma 3.2.2. *A_i is left of A_{i-1}. A_i crosses A_{i-1} at least once and only from right to left.*

Proof. First we observe that *rev*(A_{i-1}) is residual w.r.t. c_i. We find a leftmost maximum flow in G ✂ A_{i-1}, an st-planar graph, and since A_{i-1} is residual w.r.t. c_{i-1}, the flow we find is non-trivial. The leftmost of these flow paths must indeed by A_{i-1}.

A_i must cross A_{i-1} by the argument at the beginning of Section 3.2. Then, since A_i is leftmost residual, we refer to the properties guaranteed by Theorem 3.2.1, proving the second part of the lemma.

Let $A_i \otimes A_{i-1} = \{X_1, X_2, \ldots, X_{|A_i \otimes A_{i-1}|}\}$ and define $X_0 = s, X_{|A_i \otimes A_{i-1}|+1} = t$. Let x_j be any vertex in X_j. Consider the subpaths $A_i[x_j, x_{j+1}]$ and $A_{i-1}[x_j, x_{j+1}]$. The cycle $A_i[x_j, x_{j+1}] \circ rev(A_{i-1}[x_j, x_{j+1}])$ is residual, and by Invariant 1 cannot be clockwise. Therefore $A_i[x_j, x_{j+1}]$ is left of $A_{i-1}[x_j, x_{j+1}]$. □

3.2.3 Bounding the number of iterations of MaxAdaptiveFlow

We are now ready to bound the number of iterations of MaxAdaptiveFlow.

Recall that A_i, $0 \leq i \leq \rho$ is the leftmost residual path w.r.t. c_i, at iteration i of MaxAdaptiveFlow. We will use the universal cover to obtain an upper bound on the number of iterations of the algorithm.

We showed in Section 3.2.2 that A_i crosses A_{i-1} at least once and from right to left. Therefore there exists a subpath of A_i such that Y leaves A_{i-1} from the left and enters A_{i-1} from the right and there is no subpath of A_i that leaves A_{i-1} from the right and enters A_{i-1} from the left. Then by Lemma 2.4.1, $\overline{A}_{i,A_{i-1}}$ (ie. the particular lift of A_i in $\mathcal{G}_{A_{i-1}}$ that ends at $t^0_{A_{i-1}}$) must contain at least one subpath from A^{j+1}_{i-1} to A^j_{i-1} (and no A^j_{i-1}-to-A^{j+1}_{i-1} subpaths). Therefore we make progress in the following sense:

Corollary 3.2.3. *The lift of A_i in $\mathcal{G}_{A_{i-1}}$ that ends at $t^0_{A_{i-1}}$ must start at a source with a strictly positive index.*

From Lemma 2.4.4, we are able to bound how big the indices can get. Suppose s and t are separated by p faces. Embed an additional zero capacity dart across each of these faces, creating an s-to-t path P_c.

Consider $\mathcal{G}_{P_c}$. By Lemma 2.4.4, every path $\overline{A}_{P_c}$ corresponding to a simple path A (such as $A = A_i$) must start at an s_j, s.t. $|j| < |P_c|$. By Corollary 3.2.3, every $\overline{A}_{i,A_{i-1}}$ must start at a source with a strictly positive index in $\mathcal{G}_{A_{i-1}}$. Therefore by Corollary 2.4.3, if $\overline{A}_{i,P_c}$ started at a source s_j in $\mathcal{G}_{P_c}$, $\overline{A}_{i+1,P_c}$ must start at a source s_k, such that $k > j$. Therefore $i < 2|P_c| = 2p$ and:

Theorem 3.2.4. MaxAdaptiveFlow *runs in $O(np)$ time, where p is the number of faces separating s from t.*

We, however, note that $2p$ is a loose upper bound, and the realistic number of phases is likely to be many fewer.

Chapter 4: Network Flow and Shortest Paths

Maximum flow and shortest paths are linked closely in planar graphs. In 1933, Whitney discovered that a minimum cut in the primal corresponds to a shortest separating cycle in the dual [38]. In 1979, Itai and Shiloach gave an $O(n \log n)$ algorithm to find that cycle [21]. They, however, did not provide the flow function itself. Two years later, Rafael Hassin proved that a shortest-path tree in the dual rooted at f^*_∞, is sufficient to construct the maximum flow in an st-planar network, and that the construction can be done in linear time [17].

Later algorithms, including Borradaile and Klein's latest result, have diverged from these dual techniques, and rely on a primal data structure developed by Sleator and Tarjan in 1983 [35], called dynamic trees. While having desirable asymptotic running times, dynamic trees are often less efficient in practice, due to difficulty to implement and large constants.

In this section, we show that even the most general version of the planar st-flow problem can be solved with a single shortest-paths computation, by way of viewing the graph in its half-infinite cover. We start by discussing correspondence between notions of leftmost in the original graph and its cover, and show that one may obtain a max flow in the original graph, given the max leftmost flow in the cover. We then suggest running the augmenting-paths algorithm to find this leftmost flow on the universal cover, and show that it converges. This corresponds to computing shortest paths from left to right throughout the copies of the graph until the convergence point. We note that contrary to the commonly adopted intuition, we do not update the residual capacities of the network for future augmentations.

Shortest paths and planarity

Dijkstra's algorithm, published by a Dutch computer scientist Edsger Dijkstra in 1959, solves the single-source shortest-path tree problem in graphs with no negative edge lengths [8]. Dijkstra's original algorithm does not rely on min-priority queues and runs in $O(|V|^2)$ time. The $O(|E| + |V| \log |V|)$ implementation based on a min-priority queue implemented by a Fibonacci heap is due to Fredman and Tarjan [15]. Planar graphs are sparse, and this amounts to a running time of $O(n \log n)$.

There exists a linear-time algorithm for computing shortest paths in planar graphs due to Henzinger et al. that relies on planar separators [19]. While being asymptotically superior to Dijkstra's algorithm, the algorithm of Henzinger et al. has large hidden constants, and is complicated enough to have never been implemented, to our knowledge. Dijkstra's algorithm is built entirely on simple primitives and remains the most efficient in practice.

4.1 Leftmost in the cover

Let L be the leftmost path in G after removing the clockwise cycles. Let $\mathcal{G}_L^+$ be the half-infinite universal covering graph of G w.r.t. L, defined in the same way as in Section 2.4, but extending only in the positive direction.

Imagine embedding a super-source S and a super-sink T in a way that S connects to $s_i, i \geq 0$ with edges of infinite capacity in both directions, and $t_i, i \geq 0$ connect to T with edges of infinite capacity in both directions. Call this new graph $\mathcal{G}_L^{ST}$. Notice that $\mathcal{G}_L^{ST}$ is st-planar.

We begin by describing the correspondence between notions of leftmost in G and $\mathcal{G}_L^{ST}$.

Lemma 4.1.1. *Let G be clockwise-acyclic graph. The map of G in $\mathcal{G}_L^{ST}$ may contain clockwise cycles if and only if there is a counterclockwise cycle in G through t that encloses s.*

Proof. The absence of clockwise cycles in G immediately implies that there are no clockwise cycles in $\mathcal{G}_L^+$, and any such cycle in $\mathcal{G}_L^{ST}$ must use a pair of infinite capacity edges Ss_i and $s_{i'}S$, or Tt_i and $t_{i'}T$. Let C be such a cycle. There are two cases:

C uses Ss_i and $s_{i'}S$ C is of the form $Ss_i \circ P \circ s_{i'}S$, where $i' > i$, and P is a path. P is a clockwise cycle in G through s, and since $i' > i$, must be clockwise. Contradiction.

C uses Tt_i and $t_{i'}T$ C is of the form $t_{i'}T \circ P \circ Tt_i$, where $i > i'$, and P is a path. P maps to a counterclockwise cycle through t in G that encloses s.

□

Let $C^{\circlearrowleft}$ denote a counterclockwise cycle in G through t that encloses s. If there are multiple such cycles sharing a boundary, we choose the smallest one (enclosing the fewest faces). $C^{\circlearrowleft}$ maps to an infinite right-to-left path P_∞ in $\mathcal{G}_L^{ST}$. Let $C_i^{\circlearrowleft}$ be the clockwise cycle in $\mathcal{G}_L^{ST}$ of the following form: $C_i^{\circlearrowleft} = P_\infty[t_{i+1}, t_i] \circ t_iT \circ Tt_{i+1}$. Super-imposing the clockwise cycles $C_i^{\circlearrowleft}, C_{i+1}^{\circlearrowleft}, \ldots$ gives an *infinite* clockwise cycle $\mathcal{C}_i^{\circlearrowleft} = P_\infty[\cdot, t_i] \circ t_iT \circ Tt_\infty$. See Fig. 4.1. We refer to $P_\infty[\cdot, t_j]$ as an ∞-to-t_j path for some sink t_j.

Recall that a flow $\mathbf{f}$ is leftmost in a graph G, if there are no clockwise residual cycles in G w.r.t. $\mathbf{f}$. The following lemma gives an equivalent condition for the flow to be leftmost in the cover of G.

Lemma 4.1.2. *A flow in $\mathcal{G}_L^{ST}$ is leftmost iff there are no clockwise residual cycles in G, and no ∞-to-t_j residual paths in $\mathcal{G}_L^{ST}$.*

Proof. Let P_∞ be an arbitrary ∞-to-t_j path, with $C^{\circlearrowleft}$ being its corresponding cycle in G. Since $t_jT/Tt_{j'}$ are of infinite capacity for any $j, j' > 0$, saturating P_∞ saturates its corresponding infinite clockwise residual cycle $\mathcal{C}$. We now show that saturating P_∞ does not introduce clockwise residual cycles in $\mathcal{G}_L$. Assume the contrary, and let such a cycle be C. Since C must use a subpath of $rev(P_\infty)$, and is clockwise, it follows that C must

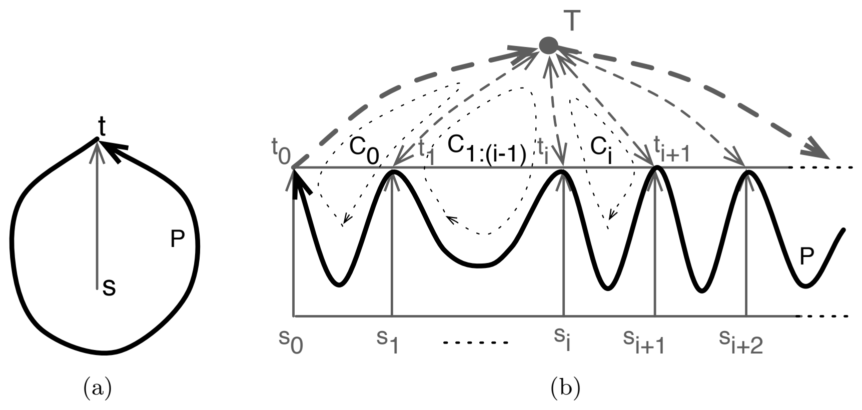

Figure 4.1: (a) P is a counterclockwise cycle through t around s in G. (b) The universal cover of G is given in grey, with dashed infinite-capacity edges. P maps to an ∞-to-t_0 path (black). $C_i = P[t_{i+1}, t_i] \circ t_iT \circ Tt_{i+1}$, $i \geq 0$ is a clockwise cycle (dotted). The bold path corresponds to the super-imposed infinite cycle $\mathcal{C} = P[\infty, t_0] \circ t_0T \circ Tt_\infty$.

contain a u-to-v residual path R, s.t. u comes before v on P_∞, and R is below P_∞ in $\mathcal{G}_L$. Then, $P_\infty[\cdot, u] \circ R \circ P_\infty[v, t_j]$ maps to a residual counterclockwise cycle in G that is smaller than and shares a boundary with $C^{\circlearrowleft}$, contradicting the choice of $C^{\circlearrowleft}$. □

4.2 Maximum flow: there and back again

Imagine mapping the maximum leftmost flow $\mathbf{f}$ in G into $\mathcal{G}_L^{ST}$. A natural way to do this is to lift the non-crossing path decomposition of $\mathbf{f}$ in G into paths in $\mathcal{G}_L$ and, since $\mathcal{G}_L^{ST}$ is only half-infinite, cutting the resulting map at L_0. Let the resulting flow assignment be $\mathcal{F}^+$.

Lemma 4.2.1. *$\mathcal{F}^+$ is a maximum preflow in $\mathcal{G}_L^{ST}$. $\mathcal{F}^+$ may be converted into a maximum flow $\mathcal{F}$ by modifying flow in at most p copies, where p is the number of faces separating s from t in G.*

Proof. Every vertex, except for those on L_0, is balanced in $\mathcal{G}_L^{ST}$. Since $\mathbf{f}$ is leftmost, by Theorem 3.2.1 there are no flow paths that cross L_0 from the left, and L_0 only contains vertices with excess flow. The cut-cycle induced by $\mathbf{f}$ in G^* corresponds to an infinite path in $\mathcal{G}_L^{ST*}$ that guarantees for no s_i-to-t_j residual paths. It remains to show that there are no v^+-to-t_i residual paths in $\mathcal{G}_L^{ST}$, where v^+ is an excess vertex on L_0. Assume the contrary. A vertex can only have excess in $\mathcal{G}_L^{ST}$, if it is on an s-to-t flow path in G. Let R be a v^+-to-t_i residual path in $\mathcal{G}_L$, and let F be the lift of the flow path through v^+ to some sink t_{-k}. The path $rev(F[t_{-k}, v^+]) \circ R$ is residual, and since $-k < i$, is oriented left-to-right, mapping to a clockwise residual cycle that passes through v and t in G. Contradiction.

Section 2.2 describes the procedure of converting a maximum preflow[1] into a maximum flow in linear time. Since every excess vertex on L_0 must be on an s-to-t flow path in G,

[1]A maximum preflow is a pseudoflow satisfying the invariant of no $S \cup V^+$-to-$T \cup V^-$ residual paths, for which V^- is empty.

and flow paths are simple, it follows from Lemma 2.4.4 on $\mathcal{G}_{P_c}^{ST}$ that the number of copies such flow paths travel is at most p, and the conversion would modify at most p copies [2]. □

By Lemma 4.1.2, $\mathcal{F}$ may not be leftmost, as there may be ∞-to-t_j residual paths in $\mathcal{G}_L^{ST}$ w.r.t. $\mathcal{F}$. Saturating the ∞-to-t_j residual paths in $\mathcal{G}_L^{ST}$ w.r.t. $\mathcal{F}$ produces a flow assignment $\mathcal{F}_\lambda$ that is leftmost in $\mathcal{G}_L^{ST}$. While $\mathcal{F}$ is balanced at every vertex in $\mathcal{G}_L^+$, $\mathcal{F}_\lambda$ is a pseudoflow in a particular copy G_i: there may be both excess and deficit vertices on L_i and L_{i+1}, due to saturating the ∞-to-t_j paths. Let k be the first index, for which the flow assignment w.r.t. $\mathcal{F}^+$ in G_k is maximum in G ($k \leq p+1$). Let the portion of $\mathcal{G}_L^{ST}$ between L_0 and L_{k+1} be $\mathcal{G}_{L_{0:k+1}}^{ST}$. The only misbalanced vertices in $\mathcal{G}_{L_{0:k+1}}^{ST}$ w.r.t. $\mathcal{F}_\lambda$ are on L_{k+1}. Note that all misbalanced vertices are on ∞-to-t_j flow paths. There are two phases: converting $\mathcal{F}_\lambda$ into a maximum postflow $\mathcal{F}^-$. Then, converting $\mathcal{F}^-$ into a maximum flow in G_k, and, consequentially, in G.

Lemma 4.2.2. *The pseudoflow $\mathcal{F}_\lambda$ in $\mathcal{G}_{L_{0:k+1}}^{ST}$ can be converted into a maximum flow $\overline{\mathcal{F}}$ in $\mathcal{G}_{L_{0:k+1}}^{ST}$.*

Proof. We focus on the flow $\mathcal{F}_\lambda - \mathcal{F}$ induced by augmenting the ∞-to-t_j residual paths. Since the flow is acyclic and all excess and deficit nodes are on flow paths that do not start at the sources, we may apply the procedure from Section 2.2, stopping when there are no more excess nodes. This produces a postflow $\mathcal{F}^-$.

We now show that $\mathcal{F}^-$ is maximum. Let P_∞ be an arbitrary ∞-to-t_j residual path w.r.t. $\mathcal{F}$. Since every subpath of P_∞ must be residual w.r.t. $\mathcal{F}$, the minimum cut w.r.t. $\mathcal{F}$ must be below P_∞ in $\mathcal{G}_L^{ST}$. Since all deficit nodes v^- are on some P_∞, it follows that there are no s-to-v^- residual paths in $\mathcal{G}_{L_{0:k+1}}^{ST}$, and $\mathcal{F}^-$ is maximum.

Maximum flow $\overline{\mathcal{F}}$ in $\mathcal{G}_{L_{0:k+1}}^{ST}$ may be obtained from maximum postflow[3] $\mathcal{F}^-$ by applying the procedure of Section 2.2 directly. □

The ST-cut induced by $\mathcal{F}^+$ persists in $\overline{\mathcal{F}}$. Therefore, by Lemma 4.2.1, G_{p+1} contains the maximum flow of G.

4.3 Convergence

We now suggest running the leftmost augmenting-paths algorithm on $\mathcal{G}_L^{ST}$. $\mathcal{G}_L^{ST}$ is st-planar, and the leftmost-path algorithm produces the leftmost max flow $\mathcal{F}_\lambda$. By Lemma 4.2.2, this flow maps to a pseudoflow in G after modifying at most p copies of the graph, and this pseudoflow can be converted into a max flow. It remains to show that the augmenting-paths algorithm on the infinite graph converges in a finite number of iterations.

The usual stopping condition for an augmenting-paths algorithm is to terminate when there are no more residual paths from the source to the sink. The universal cover is infinite,

[2]Recall that P_c is a zero-length path of p darts corresponding to the shortest path of faces from s to t.

[3]A maximum postflow is a pseudoflow satisfying the invariant of no $S \cup V^+$-to-$T \cup V^-$ residual paths, for which V^+ is empty.

and so are the clones of s-to-t residual paths. We call the flow assignment $\mathbf{f}_i$ in G_i is *final* if all paths to t_i have been augmented (alternatively, when running Dijkstra's algorithm in the dual graph, if all faces adjacent to t_i^* have been popped off the priority queue). We note that since $\mathcal{G}_L^{ST}$ is st-planar, flow never gets removed from edges, and $\mathbf{f}_i$ persists through future augmentations to sinks $i+1, i+2, \dots$

We say that the algorithms has *converged at G_k*, if the maximum flow in G can be recovered from the flow assignment in G_k. k is then the *convergence index.* Section 4.2 shows that an upper bound for the convergence index is $p+1$. It follows that:

Theorem 4.3.1. *The final flow assignment in G_{p+1} can be converted to a maximum flow in G.*

4.3.1 When is the flow final?

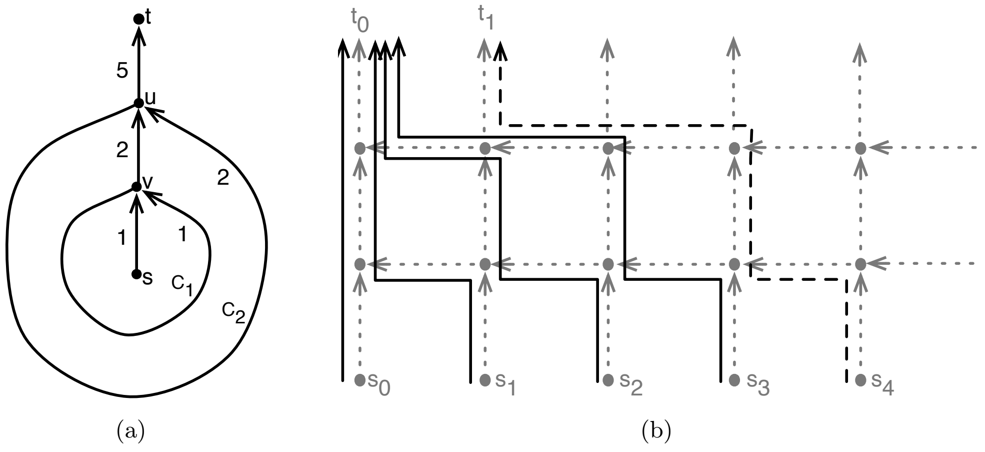

Figure 4.2: (a) G contains two counterclockwise cycles around s: C_1 and C_2, of capacities 1 and 2 respectively. Upon routing 1 unit of flow on the s-to-t path, $C_1 \circ vt \circ tv$ and $C_2 \circ ut \circ tu$ are residual cycles through t around s. (b) G's half-infinite universal cover is given with grey dotted edges. Solid paths correspond to augmentations to t_0; the dashed path is an augmentation to t_1. Convergence occurs after s_3, instead of s_0.

It is left to show that we can obtain the final flow in G_{p+1} in a finite amount of time. For this, we must bound the index j, s.t. augmenting the last s_{p+1+j}-to-t_{p+1} residual path P_{p+1+j}^{p+1} isolates the sink t_{p+1}. If P_{p+1+j}^{p+1} is simple, j must be bounded by p, by Lemma 2.4.4. However, P_{p+1+j}^{p+1} need not be simple, due to the possible counterclockwise residual cycles through t around s in G w.r.t. the flow (Fig. 4.2). Then, the number of copies G_i travelled by P_{p+1+j}^{p+1} may be offseted by at most U: the sum of the residual capacities of such counterclockwise residual cycles. This brings the bound for j to $U+p$, and:

Theorem 4.3.2. *The augmenting-paths algorithm converges in the universal cover of a graph G in $O(p+U)$ copies, where p is the number of faces separating s from t, and U is the sum of capacities in G.*

4.4 A note on implementation

The modified cover graph, $\mathcal{G}_L^{ST}$ is st-planar. Therefore augmenting-paths can be implemented with a single shortest-path tree computation in the dual, rooted at f_∞^*, growing from left to right. This tree may be computed with Dijkstra's algorithm that stops once the face adjacent to t_{p+1} has been popped off the priority queue. This computation produces a pseudoflow. Converting this pseudoflow into a maximum postflow, and the maximum postflow into a maximum flow can both be done with the procedure from Section 2.2 that may be implemented with a single traversal of the graph's topological ordering.

Chapter 5: Conclusion

In Chapter 3, we proposed a novel, yet conceptually simple algorithm for finding maximum st-flow in directed planar graphs. Using Dijkstra's algorithm at each iteration yields an efficient and implementable algorithm for finding max flow that is especially fast for graphs in which the source and the sink are separated by a sub-logarithmic number of faces. The number of faces separating s and t is a natural parameter to consider for designing adaptive algorithms, especially in the light of Kaplan and Nussbaum's recent result, and investigating it further seems hopeful.

Chapter 4 provides an interesting structural correspondence between augmenting paths in a half-infinite universal cover of a planar graph and its maximum st-flow. While being non-intuitive, this correspondence generalizes the st-planar case and affirms the historical interweaving of flows and single-source dual shortest paths, even in the most general planar graphs. The analysis of Chapter 4 also implies a peculiar property: contrary to the first lesson about max flow algorithms, the capacity updating step is not necessary (given a finite number of copies).

5.1 Discussion and future work

Tightening upper bounds. The number of iterations of MaxAdaptiveFlow and the number of copies travelled in the augmenting-paths approach on the universal cover are both loosely bounded by p on merely geometrical grounds. It may be possible to tighten this bound by investigating the structure of the solutions further. For example, when running Dijkstra's on the dual graph, instead of stopping at a fixed point of t^*_{p+1}, it may be possible to pose a convergence condition to check for at each t^*_i. It is also likely for it to be possible to reduce the generous upper bound U from Theorem 4.3.2.

Toward an $O(n \log p)$ algorithm for max flow. In general, adaptive algorithms have a tendency to follow the progression pointed out in Chapter 3: combining an existing $O(n \log n)$-time algorithm with an $O(np)$-time adaptive algorithm yields an $O(n \log p)$-time adaptive algorithm, and sometimes even produces an algorithm of a running time with an entropy factor. However, in this thesis we did not utilize the techniques of Borradaile and Klein's $O(n \log n)$ algorithm. Specifically, their Unusability Theorem is a strong structural result that may offer significant asymptotic improvements. Finding a way to integrate it into the presented analysis may reduce the running time of MaxAdaptiveFlow from $O(np)$ to $O(n \log p)$, and give a max flow algorithm in directed planar graphs that beats the current best time of $O(n \log n)$, and matches the running time of Kaplan and Nussbaum's algorithm for min cut in undirected planar graphs.

Implementing the $O(n \log n)$ max flow algorithm with priority queues. Utilizing Unusability in the presented paradigm of computing shortest paths on the universal cover may lead to an $O(n \log n)$-time algorithm that relies entirely on simple primitives. In order for Unusability to apply, one would need to non-trivially integrate the step of updating residual capacities in consequent copies into the execution of Dijkstra's algorithm. Such an algorithm would break the spell of asymptotically efficient, but practically challenging algorithms, and serve as a state-of-the-art algorithm for finding maximum flow that can be implemented by a computer science amateur.

Bibliography

[1] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *FOCS*, 2009.

[2] R. Ahuja, R. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.

[3] G. Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems.* PhD thesis, Brown University, 2008.

[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] G. Borradaile and P. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *Journal of the ACM*, 56(2):1–30, 2009.

[6] G. Borradaile, P. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. Technical Report 1105:2228, arXiv, 2011. To appear in FOCS.

[7] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Heidelberg, third edition, 2005.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 10.1007/BF01386390.

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

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

[11] J. Erickson. Maximum flows and parametric shortest paths in planar graphs. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 794–804, 2010.

[12] 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.

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

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

[15] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

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

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

[20] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

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

[22] G. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 313–322, New York, NY, USA, 2011. ACM.

[23] 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.

[24] Donald B. Johnson and Shankar M. Venkatesan. Partition of planar flow networks. In *SFCS '83: Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 259–264, Washington, DC, USA, 1983. IEEE Computer Society.

[25] H. Kaplan and Y. Nussbaum. Minimum st-cut in undirected planar graphs when the source and the sink are close. In *Proceedings of the 28th Int. Symp. Theoretical Aspects Comput. Sci.*, pages 117–128, 2011.

[26] Haim Kaplan and Yahav Nussbaum. Maximum flow in directed planar graphs with vertex capacities. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 397–407. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-04128-0-36.

[27] A. V. Karzanov. Determining the maximal flow in a network with the method of preows. *Soviet Math Dokl.*, 15:1277–1280, 1974.

[28] 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.

[29] V. King, S. Rao, and R. Tarjan. A faster deterministic maximum flow algorithm. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, SODA '92, pages 157–164, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[30] V. King, S. Rao, and R. Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447 – 474, 1994.

[31] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

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

[33] 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.

[34] Alexander Schrijver. On the history of combinatorial optimization (till 1960).

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

[36] E.H. Spanier. *Algebraic Topology*. McGraw-Hill series in higher mathematics. Springer, 1994.

[37] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

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