Minimum $s-t$ cut in undirected planar graphs when the source and the sink are close*

Haim Kaplan[1] **and Yahav Nussbaum**[1]

1 **The Blavatnik School of Computer Science, Tel Aviv University, 69978 Tel Aviv, Israel**
{haimk,yahav.nussbaum}@cs.tau.ac.il

Abstract

Consider the minimum $s-t$ cut problem in an embedded undirected planar graph. Let p be the minimum number of faces that a curve from s to t passes through. If $p=1$, that is, the vertices s and t are on the boundary of the same face, then the minimum cut can be found in $O(n)$ time. For general planar graphs this cut can be found in $O(n\log n)$ time. We unify these results and give an $O(n\log p)$ time algorithm. We use cut-cycles to obtain the value of the minimum cut, and study the structure of these cycles to get an efficient algorithm.

1998 ACM Subject Classification G.2.2 Graph algorithms; F.2.2 Computations on discrete structures

Keywords and phrases planar graph; minimum cut; shortest path; cut cycle

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.117

1 Introduction

The minimum $s-t$ cut problem is a well-studied problems with applications in many fields. By the Max-Flow Min-Cut Theorem [4], the value of the minimum $s-t$ cut is the same as the value of the maximum $s-t$ flow, and a minimum cut can be easily obtained from a maximum flow.

A planar graph is a graph that has an embedding in the plane such that no pair of edges cross each other. General maximum flow algorithms can solve the maximum flow and the minimum cut problems on planar graphs with n vertices and $O(n)$ edges in $O(n^2\log n)$ time. On the other hand, algorithms that take advantage of the structure of the planar embedding of the graph can find the minimum cut and the maximum flow in $O(n\log n)$ time (see below). The history of the maximum problem on planar graphs is surveyed in [2]. In this paper we focus on undirected planar graphs.

Itai and Shiloach [11] used the correspondence between an $s-t$ cut and a cycle in the dual planar graph (see Sect. 2) separating the dual face s^* that correspond to s from the dual face t^* that corresponds to t. Such a cycle is called a *cut-cycle*. Itai and Shiloach gave an $O(n^2\log n)$ time algorithm for finding a minimum cut using cut-cycles in undirected planar graphs. Reif [18] improved the time bound of the algorithm to $O(n\log^2 n)$ using a divide-and-conquer approach. Frederickson [5] improved the time bound of the last algorithm to $O(n\log n)$ by providing a faster shortest paths algorithm. Hassin and Johnson [8] completed the picture by showing how to find also the maximum flow within the same time bound. The

* This research was partially supported by the United States - Israel Binational Science Foundation, project number 2006204.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 117–128
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

cut-cycle approach was also used by Johnson [13] to get a parallel algorithm for maximum flow in directed planar graphs in $O(\log^3 n)$ time using $O(n^4)$ processors or in $O(\log^2 n)$ time using $O(n^6)$ processors. However, the sequential time bound of [13] is not better than the time bound of previous algorithms for the problem.

The $O(n \log n)$ time bound of [5, 8, 18] is the best time bound known for undirected planar graphs. This bound was matched for directed planar graphs by the maximum flow algorithm of Borradaile and Klein [2], using a different approach. The latter algorithm was simplified by Schmidt et al. [17] and Erickson [3]. However, the asymptotic running time of these simplified versions remains $O(n \log n)$.

Consider a planar graph embedded in the plane. Let p be the minimum number of faces that a curve from s to t passes through (the curve might go through vertices and edges of G). The graph is st-planar if and only if $p = 1$. This parameter p was first introduced by Itai and Shiloach [11] who gave an $O(np \log n)$ time algorithm for maximum flow in directed planar graphs if the value of the flow is known. Johnson and Venkatesan [14] gave an $O(np \log n)$ algorithm, without knowing the value of the flow in advance. The algorithm of [14] has two bottlenecks, the first one is the computation of maximum flow in st-planar graphs, and the second is transforming a planar flow into an acyclic flow. The first bottleneck was addressed by Henzinger et al. [9] and the second by Kaplan and Nussbaum [16]. Hence, we get an $O(np)$ time algorithm for flow in planar graphs, which is faster than the $O(n \log n)$ time algorithm for $p = o(\log n)$.

There is a well-known algorithm (see for example [10, Chap. 10]) for the minimum cut problem in directed st-planar graphs using a shortest path algorithm in the dual planar graph. With the shortest path algorithm of [9] this takes $O(n)$ time. Hassin [7] extended this algorithm to a maximum flow algorithm with the same time bound.

Our main result in this paper is an $O(n \log p)$ algorithm for minimum $s - t$ cut in undirected planar graphs. This algorithm runs in $O(n)$ time when the graph is st-planar ($p = 1$), matching [7], and in $O(n \log n)$ time for general undirected planar graphs, matching [5, 8, 18]. In general, p might be $\Theta(n)$, but p is small when s and t reside on the boundaries of faces which are close to each other. Our algorithm is asymptotically faster than what was previously known for any non-constant $p = o(n^\varepsilon)$ (where $\varepsilon > 0$ is constant).

Another related topological parameter q, introduced by Frederickson [6], is the minimum number of faces required to cover all vertices of the graph. It is always true that $p = O(q)$.[1] Arikati, Chaudhuri and Zaroliagis [1] gave an $O(n + q \log q)$ algorithm for the minimum $s - t$ cut problem in directed planar graphs.

We note that Janiga and Koubek [12] claimed an $O(n \log n \log p / \log \log n)$ algorithm for finding the minimum cut-cycle in directed planar graphs. Erickson [3] states that the algorithm of [12] can be implemented in $O(n \log n)$ time. However, in Appendix A we show a flaw in this algorithm.

2 Preliminaries

Let $G = (V, E)$ be an undirected simple planar graph with vertex set V and edge set E. Let $n = |V|$. Since G is planar it follows from Euler's formula that $|E| = O(n)$. We denote an edge between the vertices u and v by (u, v). We assume that the input graph is given with a

[1] Consider a curve R from s to t, and a minimum set Q of faces that cover the vertices of G. We can assume that R crosses the boundary of faces only at vertices. If two non-consecutive vertices in R are on the boundary of the same face of Q, then we can make R shorter by routing it through this face.

fixed planar embedding, in other words G is a *plane graph*.

In the graph G there are two designated vertices, the *source* s, and the *sink* t. The *capacity function* c assigns to every edge e a non-negative capacity $c(e)$.

An $s-t$ *cut*, or a *cut* for short, is a minimal set of edges S, whose removal from the graph disconnects t from s. The value of S is the total capacity of its edges, $\sum_{e\in S} c(e)$.

A *path* Q is a sequence of edges $(e_1, e_2, \ldots, e_j)$ such that $e_i = (v_i, v_{i+1})$. For $1 \leq i \leq j+1$ we say that the path Q *contains* the vertex v_i. The path Q *begins* at v_1 and *ends* at v_{j+1}. A path Q is *simple* if for every vertex v, there are at most two edges incident to v in Q. We denote by $|Q|$ the number of vertices in Q. We consider a single vertex to be a degenerate path without edges. If lengths are associated with the edges then the *length* of Q is the sum of the lengths of all the edges of Q (an edge that appears in Q multiple times contributes its length to the sum the same number of times). The *reverse of* $Q = (e_1, e_2, \ldots, e_j)$ is the path $Q^r = (e_j, e_{j-1}, \ldots, e_1)$. For two paths $Q = (e_1, e_2, \ldots, e_j)$ and $R = (d_1, d_2, \ldots, d_i)$ such that the last vertex of Q is identical to the first vertex of R, we define $Q \circ R$ to be the path $(e_1, e_2, \ldots, e_j, d_1, d_2, \ldots d_i)$.

A path that begins and ends at the same vertex is a *cycle*. We say that two cycles are identical, if they have the same sequence of edges, in the same cyclic order (it does not matter which vertex we pick as the first/last).

A *flower-cycle* is a cycle with a special structure defined as follows. Let Q be a simple path whose last vertex is w, let B be a simple cycle that begins and ends at w. Assume that B and Q do not share any vertex except w. Then, the cycle $C = Q \circ B \circ Q^r$ is a *flower-cycle*. We call the cycle B the *blossom* of the flower-cycle C, and we call the cycle $S = Q \circ Q^r$ the *stem* of C.

For the plane graph G, the *dual graph* G^* is defined in the following way. The vertex set of G^* is the set of faces of G. Two vertices of G^* are adjacent if and only if the boundaries of the corresponding faces share an edge. The graphs G and G^* share an embedding in the plane, such that for every vertex v of G there is a unique dual face v^* in G^* that contains v, and for every vertex x^* of G^* the face x of G contains x^*. For an edge e of G, there is a single dual edge e^* that crosses e in the shared embedding of G and G^*. The capacity of an edge e of G is interpreted in G^* as the *length* of e^*. We fix our embedding such that s^* is the infinite face of G^*.[2] (See Fig. 1).

Consider a simple cycle C in G^*. The cycle C separates the plane into two connected regions. One of the regions, which contains s^*, is *outside* C and the other is *inside* C, the edges and the vertices of C are contained in both regions. We say that an edge e or a vertex v is *strictly inside* (resp. *strictly outside*) C, if it is inside (resp. outside) C, but does not belong to C.

Let C be a simple cycle in G^* such that the face t^* is inside C. The face s^* must be strictly outside C, so the cycle separates s from t in the plane. We call such C a *cut-cycle*. The edges of G that are crossed by the edges of C form an $s-t$ cut. These edges are exactly the edges whose duals are in C. The value of the cut is the same as the length of the cut-cycle C. Therefore, we get that the value of the minimum cut is the same as the value of the shortest cut-cycle [11]. (See Fig. 1).

Let Q be any path from a vertex on the boundary of s^* to a vertex on the boundary of t^*. Let $x_1^*, \ldots, x_q^*$ be the vertices of Q where x_1^* is incident to s^* and x_q^* is incident to t^*. We say that an edge (x_i^*, y^*) *emanates left* from Q at x_i^*, if when traversing Q from x_1^* to x_q^*,

[2] Most recent papers, e.g. [2, 3], fix t to be on the boundary of the infinite face of G, we choose to fix s on the boundary of the infinite face to be consistent with the previous cut-cycle algorithms [8, 11, 18].

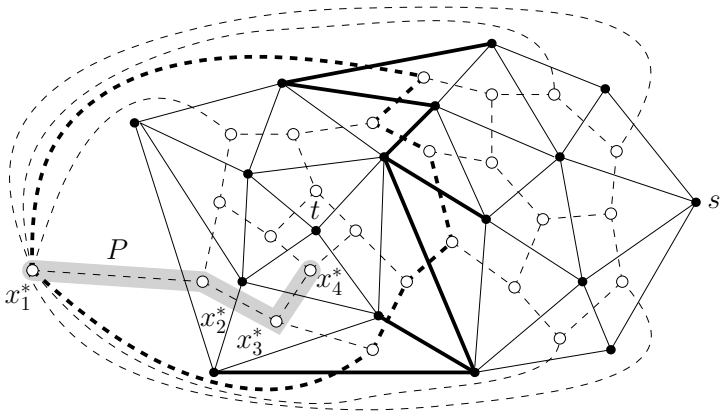

Figure 1 A planar graph and its dual. The vertices of G are *dots* and its edges are *solid*. The vertices of G^* are *circles* and its edges are *dashed*. The infinite face of G^* is s^*. The path P from s^* to t^* is *shaded*. The *bold* edges are an $s-t$ cut in G, their dual edges are a cut-cycle that contains one P-left edge incident to x_1^*.

the edge is incident to x_i^* on the left side of Q, this definition is applied to edges incident to x_1^* and x_q^* by adding two dummy vertices – x_0^* inside s^* before x_1^*, and x_{q+1}^* inside t^* after x_q^*. We call an edge *Q-left* if it emanates left from Q at exactly one of its endpoint. In other words, an edge that emanates left from Q is Q-left unless it is an edge of the form (x_i^*, x_j^*) that emanates left from Q both at x_i^* and at x_j^*. *Q-right* edges are defined similarly.

Let P be a path in G^* from a vertex on the boundary of s^* to a vertex on the boundary of t^*, with minimum number of vertices. We define $p(G)$ to be the number of vertices on P. In the introduction we defined the parameter p as the minimum number of faces that a curve from s to t passes through. This parameter is equal to $p(G)$ if the curve is not allowed to go through vertices, because then every edge of P is dual to an edge that the curve from s to t crosses.

Since we want to allow the curve to contain vertices, we change G such that in the modified graph, $\tilde{G}$, there is a curve from s to t that passes through the smallest number of faces and does not contain vertices. Furthermore, this curve in $\tilde{G}$ crosses the same number of faces as the corresponding curve in G which may contain vertices. Also, the value of the minimum $s-t$ cut in $\tilde{G}$ is the same as in G. The advantage of this transformation is that in $\tilde{G}$, p is equal to $p(\tilde{G})$ – the smallest number of vertices on a path from a vertex on s^* to a vertex on t^* in $\tilde{G}^*$.

The construction of $\tilde{G}$ is as follows. We choose a curve R from s to t passing through p faces such that R goes from a face x to a face y through a vertex v only when there is no edge common to the boundaries of x and y. If R does not contain vertices then $\tilde{G} = G$. Otherwise we split every vertex on R as follows. Consider a vertex v of G that R passes through when going from a face x to a face y. In $\tilde{G}$ we split v into two vertices v' and v'' and connect them with a new edge e that separates between x and y. Every edge that was incident to v is now incident either to v' or to v'', such that G remains planar. This transformation allows R to cross the edge e instead of the vertex v. We give e a large capacity (larger than the sum of all capacities in G), so that it does not change the minimum cut.

This transformation requires knowing the curve R. We can compute R by computing a path P' from a vertex on the boundary of s^* to a vertex on the boundary of t^* with minimum number of vertices in a graph which we construct from G^* as follows. This construction is similar to a construction of Khuller and Naor [15]. For every face v^* in G^*, we add a new

vertex z^* inside the face v^* and connect z^* with edges to every vertex on the boundary of the face v^*. The new vertex z^* inside v^* allows the path P' to "jump over" the face v^* from one vertex on its boundary to another, which is equivalent to the case where the curve R passes through the vertex v of G. We also remove all the original edges from G^*. This forces R to cross edges only at incident vertices, which we can do without loss of generality. Let P' be a path with minimum number of vertices, in the graph that we constructed, from a vertex on the boundary of s^* to a vertex on the boundary of t^*. The path P' alternates between vertices of G^* and vertices that we added inside faces of G^*. For every pair of consecutive edges (x^*, z^*) and (z^*, y^*) in P', the curve R goes from the face x of G to the face y of G. If x and y share a common edge e on their boundaries in G, then R passes through e, otherwise R passes through the vertex v such that z^* is inside the face v^*.

In the rest of the paper we assume that G was preprocessed as described here (so in fact we use G to refer to $\tilde{G}$ to simplify the notation), then $p = p(G)$ is the minimum number of vertices on a path from a vertex on the boundary of s^* to a vertex on the boundary of t^*. We will denote such a path by P, we can find P in linear time using a breadth-first search on the graph G^*.

3 Finding a Minimum $s - t$ Cut

3.1 Overview

Let Π be the shortest path (path of minimum length) from a vertex incident to s^* to a vertex incident to t^*, and let $x_1^*, \ldots, x_k^*$ be the vertices on Π. Itai and Shiloach [11] observed that in an undirected planar graph, the shortest cut-cycle must cross the path Π exactly once. To exploit this observation they defined x_i^*-cycle to be a cycle containing exactly one Π-left edge and one Π-right edge, such that the Π-left edge is incident to x_i^*. Then, the minimum cut-cycle is the minimum x_i^*-cycle. Reif [18] later noticed that for every $i < j$ there is a minimum x_j^*-cycle inside a minimum x_i^*-cycle. He used this to speed up the computation of the shortest cut-cycle to $O(n \log k)$ time, using a divide-and-conquer algorithm (this is not the time bound stated by [18], but it can be obtained using techniques of [5] or the shortest-path algorithm of [9]).

If we replace Π with a path Q from s^* to t^* that is not shortest, then a cut-cycle may cross Q more than once. This makes the task of finding a shortest cut-cycle more difficult. First, there may not be a shortest cut-cycle that contains x_j^* inside a shortest cut-cycle that contains x_i^* for $i < j$, simply because x_j^* may be outside this cycle. Second, finding the shortest cut-cycle through a particular vertex is harder.

Johnson [13] showed that any cut-cycle crosses Q an odd number of times, and used this to get the parallel algorithm that we mentioned. We use the observation of Johnson that the number of crossings of a cut-cycle with Q is odd to extend the algorithm of [18] to work with any path from s^* to t^*. This way, if we take the path Q to be the path P that we defined in Sect. 2 with the minimum number of vertices from a vertex on s^* to a vertex on t^*, we get the $O(n \log p)$ time bound (recall that $p = |P|$).

Let $x_1^*, \ldots, x_p^*$ be the vertices of P, where x_1^* is incident to s^* and x_p^* is incident to t^*. First, we show how to find a shortest cut-cycle, by finding shortest simple cycles containing x_i^* that crosses P an odd number of times, for every $1 \leq i \leq p$. We characterize the structure of these cycles, and show that their structure still allows to find the shortest among them efficiently by a divide-and-conquer algorithm. Then, we show how to efficiently find the shortest cut-cycle through any particular vertex on P by computing a shortest path in a related planar graph.

3.2 Structure of shortest cut-cycles containing particular vertices

We call a cycle C an *odd-cycle* if the number of P-left edges in C is odd. The following lemma is a special case of a lemma of Johnson [13].

▶ **Lemma 1.** *Let C be a simple cycle. Then C is a cut-cycle if and only if it is an odd-cycle.*

Proof. Consider a simple cycle C and the path P. Suppose we extend P by a dummy vertex x_0^* inside s^* and by a dummy vertex x_{p+1}^* inside t^*. Assume that we walk on the plane from x_0^* to x_{p+1}^*, to the left of P and infinitesimally close to it. We start our walk outside of C (since s^* is outside of C). Each time we cross an edge of C we switch from being outside C to being inside C or vice versa. If C is a cut-cycle then x_{p+1}^* is inside C so our walk ends inside C and therefore must cross C an odd number of times. Similarly, if we cross C an odd number of times then x_{p+1}^* must be inside C and therefore C is a cut-cycle. So we conclude that C is a cut-cycle if and only if we cross C in our walk an odd number of times.

Each such crossing of the walk and C corresponds to an edge of C that emanates left from P at one of its endpoints. P-left edges emanate left from P at exactly one of their endpoints, while other edges do not emanate left from P at all, or emanate left from P at both of their endpoints. It follows that C is a cut-cycle if and only if it contains an odd number of P-left edges. ◀

We denote a *shortest odd-cycle containing the vertex x^** by $C(x^*)$. For a specific vertex x^*, the cycle $C(x^*)$ may not be simple. However, since we can decompose any cycle into simple cycles, there is a shortest odd-cycle that is simple. Moreover, any odd-cycle intersects P, so the following corollary follows from Lemma 1.

▶ **Corollary 2.** *The shortest cut-cycle is $C(x_i^*)$ for some $x_i \in P$.* ◀

Corollary 2 suggests that our definition of $C(x_i^*)$ generalizes Itai and Shiloach's definition of a minimum x_i^*-cycle. This is essential since, as we mentioned, when we replace Π by the path P, which is not a shortest path, there may be more than one P-left edge in a shortest cut-cycle.

The next lemma allows us to assume that $C(x^*)$ is a flower-cycle.

▶ **Lemma 3.** *For any vertex x^*, there is a shortest odd-cycle containing x^* that is a flower-cycle.*

Proof. The edge set of the cycle $C(x^*)$ is a union of edge sets of simple cycles, at least one of these simple cycles must be an odd-cycle. Let C be such a simple odd-cycle, and let y^* be the first vertex of C that we encounter when we traverse $C(x^*)$, starting at an occurrence of x^*.

We can decompose $C(x^*)$ into $Q \circ C \circ Q'$, where Q is a path from x^* to y^* and Q' is a path from y^* to x^*. The length of Q must be equal to the length of Q' since otherwise we can replace the longer by the reverse of the shorter and get a shorter odd-cycle through x^* (C is an odd-cycle and the sets of P-left edges of a path and its reverse are identical). Let $F = Q \circ C \circ Q^r$. The cycle F is a flower-cycle by its definition, it is also an odd-cycle, and it has the same length as $C(x^*)$. Therefore, F is a shortest odd-cycle containing x^* which is a flower-cycle. ◀

Recall that Reif [18] based his divide-and-conquer approach on the observation that inside every minimum x_i^*-cycle there is a minimum x_j^*-cycle if $i < j$. The same claim is not always true for $C(x_i^*)$ and $C(x_j^*)$. It might be possible that $C(x_i^*)$ is actually strictly inside $C(x_j^*)$. (See Fig. 2). We develop an alternative similar divide-and-conquer approach for $C(x_i^*)$.

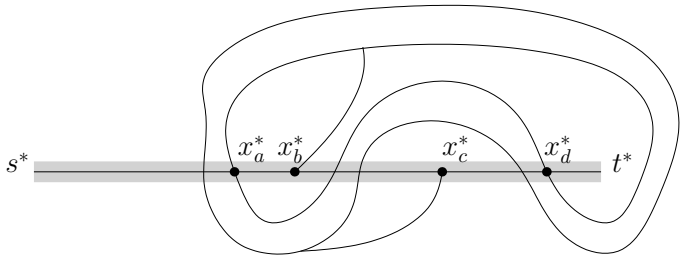

Figure 2 Structure of the flowers-cycles. $C(x_a^*)$ has an empty stem, $C(x_b^*)$ has the same blossom as $C(x_a^*)$ and stem inside the blossom, $C(x_c^*)$ has a stem outside its blossom. There are 3 P-left edges in $C(x_a^*)$ and in $C(x_c^*)$, and 5 P-left edges in $C(x_b^*)$ (the edge of the stem is counted twice). $C(x_b^*)$ is strictly inside $C(x_c^*)$ even though $b < c$. We do not have to compute any $C(x_i^*)$ for $i < d$ inside $C(x_b^*)$, even if one of them is a minimal cut-cycle we will find it for another value of i.

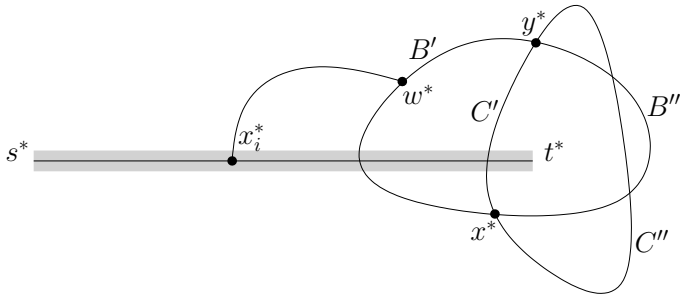

Figure 3 The shortest cut-cycle $C = C' \circ C''$ is neither inside nor outside the blossom $B = B' \circ B''$.

▶ **Lemma 4.** *Let B be the blossom of $C(x_i^*)$ for some i, and let S be the stem of $C(x_i^*)$. There is a shortest cut-cycle that is either inside B, or outside B (if B is a shortest cut-cycle then both hold).*

Proof. Assume otherwise, then for every shortest cut-cycle C, there are edges of C strictly inside B and edges of C strictly outside B.

Let w^* be the vertex common to B and S. By the minimality of $C(x_i^*)$ we may assume that $C(w^*) = B$.

Let C be a shortest cut-cycle maximizing the number of edges that it has in common with B. Let C' be a maximal subpath of C strictly inside B. Let C'' be "the rest of C" – that is, the path such that $C = C' \circ C''$. The path C' starts at a vertex x^* on B and ends at another vertex y^* on B. We split B or B^r into two parts, B' and B'', such that B' is a path from x^* to y^*, and B'' is a path from y^* to x^*, and the cycle $C' \circ B''$ is an odd-cycle. Since both B and C are odd cycles, such a decomposition of B or of B^r must exist. It follows that both $C' \circ B''$ and $B' \circ C''$ are odd-cycles. (See Fig. 3).

By our choice of C, we may assume that C' is shorter than B', as otherwise $B' \circ C''$ is not longer than C, and has more edges in common with B, contradicting the choice of C. Also, we may assume that C'' is shorter than B'', as otherwise $C' \circ B''$ is not longer than C, and has more edges in common with B, again in contradiction to the choice of C. Therefore both $C' \circ B''$ and $C'' \circ B'$ are shorter than B. The vertex w^* must be on one of the cycles $C' \circ B''$ or $B' \circ C''$, contradicting the minimality of $B = C(w^*)$. Hence the lemma follows. ◀

▶ **Lemma 5.** *Let B be the blossom of $C(x_i^*)$ for some i, and let S be the stem of $C(x_i^*)$. If B is not a shortest cut-cycle, then any shortest cut-cycle does not contain any edge incident*

to a vertex of S.

Proof. Assume that B is not a shortest cut-cycle, and let C be a shortest cut-cycle, such that there is a vertex x^* that is common to S and C. Let Q be the shortest path from x_i^* to x^*. By its definition, the path Q is not longer than half of the cycle S.

The cycle $Q \circ C \circ Q^r$ is an odd-cycle since C is an odd cycle. The cycle C is shorter than B, and the cycle $Q \circ Q^r$ is not longer than S, contradicting the minimality of $C(x_i^*)$. ◀

3.3 Divide-and-conquer algorithm

Lemma 4 gives a method for dividing the graph, in order to find a shortest cut-cycle. Consider $C(x_i^*)$ with blossom B and stem S. If B is not a shortest cut-cycle then there is a shortest cut-cycle C, such that C is either inside B or outside B. Thus, we can divide the graph into two parts, G_{in} which is the part inside B, and G_{out} which is the part outside B, and search in each of them separately. We also discard S from the subgraph containing it, which we can do by Lemma 5. This will help us to bound the depth of the recursion.

It is simpler to describe how to obtain G_{in}^* and G_{out}^* from G^*, so we do this first. We start by putting into G_{in}^* every edge and vertex of G^* that is inside B. The part of the plane strictly outside B becomes the infinite face s_{in}^*, we also set $t_{\text{in}}^* = t^*$. The graph G_{out}^* initially contains every edge and vertex of G^* that is outside B. (By our definitions of "outside" and "inside" there is a copy of B in both graphs.) In G_{out}^* we set $s_{\text{out}}^* = s^*$ and the part of the plane inside B becomes a single face which we denote by t_{out}^*.

Assume that S is not empty. Since there is a single vertex w^* that is common to S and B, S is either entirely inside B or outside B. Assume that S is outside B. According to Lemma 5 we may assume that if there is a vertex of S on a shortest cut-cycle, then it is w^*. This happens only when B is a shortest cut-cycle, in this case B is contained also in G_{in}^*. Thus, we can remove the vertices of S and all the edges adjacent to them from G_{out}^* without losing the shortest cut-cycle. Symmetrically, if S is inside B, then we remove the vertices of S and their adjacent edges from G_{in}^*. This completes the definition of G_{in}^* and G_{out}^*.

The effect of this construction on the primal graph is as follows. Consider the common embedding of G and G^*. The graph G_{in} contains all the vertices inside B and all edges with both endpoints inside B. Similarly, G_{out} contains the vertices outside B and edges with both endpoints outside B. Edges of G whose duals are in B are the edges with one endpoint outside B and one endpoint inside B. We put a copy of these edges in both graphs as follows. We add a vertex s_{in} to G_{in}, which would be the source of G_{in}, and for every edge $e = (u, v)$ such that $e^* \in B$ with v inside B, we put the edge (s_{in}, v) in G_{in} with the same capacity as of e. Similarly, we add a vertex t_{out} to G_{out}, which would be the sink of G_{out}, and for every edge $e = (u, v)$ such that $e^* \in B$ with u outside B, we put the edge (u, t_{out}) in G_{out} with the same capacity as of e. We set $t_{\text{in}} = t$ and $s_{\text{out}} = s$.

If S is not empty, then we contract every edge $e = (u, v)$ such that e^* is incident to a vertex of S, in the graph whose dual contains S. That is u and v become a single vertex, and the incidence lists are concatenated (without e) in the appropriate cyclic order. Note that the edges that we contract include all the edges on faces of G that correspond to the vertices of S in G^*. The contraction eliminates all these faces.

It is possible that the new graphs G_{in} and G_{out} are not simple. We replace a set of multiple edges of the form (s_{in}, v) or (u, t_{out}) by a single edge whose capacity is the sum of all capacities of the multiple edges. We do so to ensure that G_{in} and G_{out} remain simple. Two parallel edges e and d create a face x between them. In the dual graph, x^* is a vertex adjacent only to e^* and d^*. The effect in the dual graph of merging e and d is the removal of

x^*, and replacement of e^* and d^* by a single edge whose length is the sum of lengths of both dual edges.

Note that every vertex of G has a single copy, either in G_{in} or in G_{out}, while edges and faces of G whose duals are in B might have copies in both graphs. When we construct G_{in} and G_{out} from G, every edge of G is mapped to a single edge of G_{in} or G_{out}. However, a single edge of G_{in} or G_{out} might be mapped to more than one edge of G (due to merge of parallel edges). When we return a cut in G_{in} or G_{out} as an answer to the minimum cut problem on G, we replace every edge of G_{in} or G_{out} with all the edges of G that were mapped to it.

Now we have all the definitions required to present the divide-and-conquer algorithm for finding a minimum $s-t$ cut:

1. Find P, the path with minimum number of vertices from a vertex on the face s^* to a vertex on the face t^*. Let $p = p(G) = |P|$.
2. If $1 \leq p \leq 2$, find $C(x_i^*)$ for every $1 \leq i \leq p$, and return the shortest.
3. Otherwise, let $i = \lfloor p/2 \rfloor + 1$.
4. Find $C(x_i^*)$.
5. Construct G_{in}, G_{out} and apply the algorithm recursively to them.
6. Return the smaller between the minimum $s_{\text{in}} - t_{\text{in}}$ cut in G_{in} and the minimum $s_{\text{out}} - t_{\text{out}}$ cut in G_{out} that were computed in the previous step.

We already pointed out two differences between our algorithm and these of Reif [18] and Hassin and Johnson [8], namely using the path P instead of the shortest path Π and finding a cut-cycle that is a $C(x_i^*)$ cycle instead of a minimum x_i^*-cycle.[3] Another difference is that we do not compute $C(x_i^*)$ for every x_i^* in the original path P from s^* to t^*. Since $C(x_i^*)$ may cross P multiple number of times it is possible, for example, that for some $j > i$, x_j^* is on the boundary of s_{in}^*, and so we do not need to apply our algorithm in G_{in} for $x_{i'}^*$ such that $i' < j$ (see Fig. 2). A symmetric claim is true for G_{out}. For this reason, we compute the path P in the first step of each recursive call. This is easy to do in time linear in the size of the input graph, by using breadth-first search on the dual graph.

The correctness of our algorithm follows from Lemma 4 and Lemma 5. In order to get the desired $O(n \log p)$ time bound, we show that the depth of the recursion is $\lceil \log p \rceil + 1$ and that it is possible to find $C(x_i^*)$ in $O(n)$ time.

To bound the depth of the recursion we show that $p(G_{\text{in}}), p(G_{\text{out}}) \leq \lfloor p/2 \rfloor + 1$.[4]

First, assume that S is empty, that is, $x_i^* \in B$. The copy of the vertex x_i^* in the graph G_{in}^* is on the boundary of s_{in}^*. The subpath $(x_i^*, x_{i+1}^*, \ldots, x_p^*)$ of P is not necessarily in G_{in}^*, but G_{in}^* must contain a suffix of this subpath that starts with some x_j^*, $j \geq i$, that is on the boundary of s_{in}^*. This implies that $p(G_{\text{in}}) \leq \lfloor p/2 \rfloor + 1$. Similarly, the copy of the vertex x_i^* in G_{out}^* is on the boundary of t_{out}^* so there is a prefix $(x_1^*, \ldots, x_{j'-1}^*, x_{j'}^*)$, $j' \leq i$ of P, such that $x_{j'}^*$ is on the boundary of t_{out}^* in G_{out}^*. This shows that $p(G_{\text{out}}) \leq \lfloor p/2 \rfloor + 1$.

Now assume that S is not empty and that it is outside B. Let w^* be the vertex common to S and B. Since B is an odd-cycle it must contain a P-left edge. Since x_i^* is outside B, at

[3] Another minor change from the algorithm of [18] is in the base of the recursion (Step 2), this correction was suggested by [8].

[4] Consider a binary representation b of p. We obtain a binary representation of an upper bound on the new value of p following a recursive call, by shifting b one position to the right and adding one to the result. It follows that the number of bits in the representation of the upper bound decreases by one every step but can increase by one at most once, so the depth of the recursion is at most the number of bits in b plus one.

least one P-left edge of B is incident to x^*_j for some $j \geq i$, so the proof that $p(G_{\text{in}}) \leq \lfloor p/2 \rfloor + 1$ does not change. We removed the vertices of S and their incident edges from G^*_{out}. Before this removal, w^* was on the boundary of t^*_{out}, so after the removal, every vertex that was adjacent to a vertex of S, is on the boundary of t^*_{out}. The vertex x^*_i is in S, so there must be a vertex $x^*_{i'}$ with $i' < i$ that was adjacent to vertex in S, and is now on the boundary of t^*_{out}. We get that there is a prefix $(x^*_1, \ldots, x^*_{j'-1}, x^*_{j'})$, $j' \leq i'$, of P in G^*_{out} which shows that $p(G_{\text{out}}) \leq \lfloor p/2 \rfloor$. The proof for the case where S is inside B is symmetric.

We conclude that the depth of the recursion is at most $\lceil \log p \rceil + 1$.

3.4 Finding a shortest odd-cycle containing a particular vertex

Now we show how to find $C(x^*_i)$, a shortest odd-cycle that is a flower-cycle containing a specific vertex x^*_i of P. We do so in $O(n)$ time using the following construction.

We create a new planar graph H that contains two modified copies of G^* as follows. For every vertex x^* in G^* we create two copies in H, x^0 and x^1. For every edge (x^*, y^*) that is not a P-left edge we create two copies (x^0, y^0) and (x^1, y^1) in H. Last, for every P-left edge (x^*_j, y^*) we create two copies (x^0_j, y^1) and (x^1_j, y^0). We denote the set that contains the vertices x^0, the edges (x^0, y^0) and the edges (x^0_j, y^1) by H^0, and the set of other vertices and edges by H^1. A path from x^* to y^* in G^* corresponds to a path from x^0 to y^0 or a path from x^0 to y^1 in H. We denote the image in H of a path Q in G^* by $h(Q)$.

Let Q be a cycle in G^*, fix x^* to be the first vertex of Q. The image $h(Q)$ begins with x^0, which is in H^0. Assume that we traverse $h(Q)$, starting at x^0. Every time that $h(Q)$ goes through an image of a P-left edge, it jumps from H^0 to H^1 or vice versa. Therefore, Q is an odd-cycle if and only if $h(Q)$ ends at x^1.

For a vertex x^*_i of P, we find a shortest odd-cycle through x^*_i by finding a shortest path R from x^0_i to x^1_i in H. The cycle $h^{-1}(R)$ is a shortest odd-cycle containing x^*_i.

Although by Lemma 3 there is a shortest odd cycle through x^*_i which is a flower-cycle the cycle $h^{-1}(R)$ may not be a flower-cycle. We can convert it to a flower-cycle as suggested by the proof of Lemma 3. Let y^* the first vertex that repeats twice on $h^{-1}(R)$, and let C be the path in $h^{-1}(R)$ between the first two occurrences of y^*. If C is not and odd-cycle (this may happen only if the length of C is 0), then remove the edges of C from $h^{-1}(R)$ and repeat the process until a simple odd-cycle is found. Let Q be the prefix of $h^{-1}(R)$ that ends at y^*. Finally, let $C(x^*_i) = Q \circ C \circ Q^r$.

The construction of H takes $O(n)$ time, and finding R takes $O(n)$ time using the algorithm of Henzinger et al. [9]. Replacing $h^{-1}(R)$ with a flower-cycle takes $O(n)$ time as well. We conclude that a single recursive application of our algorithm is linear in the size of the graph it works on.

3.5 Running time

The running time analysis for our main algorithm is similar to that of Reif [18] or Hassin and Johnson [8], we use here the one of [8]. As we showed, the depth of the recursion tree is at most $\lceil \log p \rceil + 1$. In each recursive call, when we split G into G_{in} and G_{out}, we add two vertices to the graphs. Therefore, at the ℓ^{th} level of the recursion we have at most $n + 2^\ell$ vertices. The time bound at each level of the recursion is linear in the number of vertices at the level so the total running time of our algorithm is $O\left(\sum_{\ell=0}^{\lceil \log p \rceil} \left(n + 2^\ell\right)\right) = O(n \log p)$.

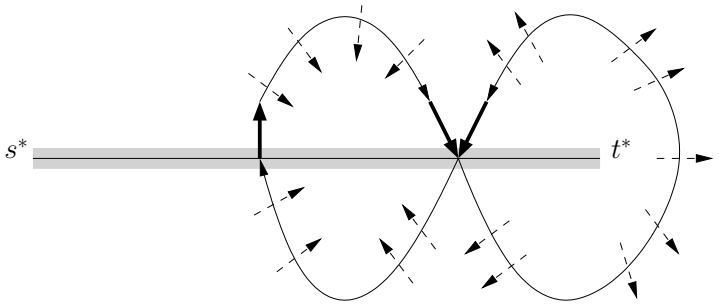

Figure 4 Counterexample to the algorithm of [12] (see [13, Fig. 7]). The directed cycle contains three P-left arcs (*bold*), such that the arc farthest from t^* is oriented away from P. However, the cycle is not simple and it wraps around t^* in the wrong direction. The direction of the primal arcs that correspond to the cycle is indicated with *dashed arrows*.

A The Algorithm of [12]

Janiga and Koubek [12] presented an $O(n \log n \log p / \log \log n)$ algorithm for the minimum $s-t$ cut problem in directed planar graphs, based on the cut-cycles approach. Erickson [3] states that this algorithm can be implemented in $O(n \log n)$ time. In this appendix we show that there is a mistake in the algorithm of [12].

To deal with directed graphs we have to extend the definitions from Sect. 2. First, in a directed graph, two anti-parallel arcs, $(u \to v)$ and $(v \to u)$ may have different capacities. Second, the dual graph G^* is also directed. The dual of an arc $d = (u \to v)$ is the arc in G^* which is directed from the dual vertex of the face on the *right side* of d to the dual vertex of the face on the *left side* of d.[5] The path P that the algorithm of [12] uses is a *directed path* with minimum number of vertices from a vertex on the boundary of s^* to a vertex on the boundary of t^*.

In the directed graph G^*, a cut-cycle is dual to a cut in G, if and only if it is oriented *clockwise* around t^* in the shared embedding of G and G^* in the plane [13].

Janiga and Koubek [12, Sect. 3] look for the minimum cut by computing the shortest cycle in G^* that its P-left arc farthest from t^* is oriented away from P (that is, the P-left arc which is adjacent to x_i^* for the minimum i is oriented $(x_i^* \to y)$), and that crosses P an odd number of times. If this shortest cycle is *simple*, then it is a cut-cycle oriented clockwise around t^* [13] (the proof is similar to Lemma 1). However, it is possible that the shortest cycle that fulfills these requirements is not simple. In this case, the cycle may be oriented counterclockwise around t^*, and therefore it would not be dual to a cut. Figure 4 shows an example of such case, which was given by Johnson [13, Fig. 7]. Since algorithm *Cycle2* of [12, p. 43] fails to check whether the path it finds it simple or not, the algorithm of [12] will not find a correct solution in this case.

References

1 Arikati, S.R., Chaudhuri, S., Zaroliagis, C.D.: All-pairs min-cut in sparse networks. *J. Algorithms* 29, 82–100 (1998)

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

[5] Some papers, e.g. [2, 3], use the opposite orientation.

3 Erickson, J.: Maximum flows and parametric shortest paths in planar graphs. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 794–804 (2010)

4 Ford, L.R., Fulkerson, D.R.: *Flows in Networks.* Princeton University Press, New Jersey (1962)

5 Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.* 16, 1004–1022 (1987)

6 Frederickson, G.N.: Using cellular graph embeddings in solving all pairs shortest path problems. *J. Algorithms* 19, 45–85 (1995)

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

8 Hassin, R., Johnson, D.B.: An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comput.* 14, 612–624 (1985)

9 Henzinger, M.R., Klein, P., Rao, S., Subramania, S.: Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55, 3–23 (1997)

10 Hu, T.C.: *Integer Programming and Network Flows.* Addison-Wesley, MA (1969)

11 Itai, A., Shiloach, Y.: Maximum flow in planar networks. *SIAM J. Comput.* 8, 135–150 (1979)

12 Janiga, L., Koubek, V.: Minimum cut in directed planar networks. *Kybernetika* 28, 37–49 (1992)

13 Johnson, D.B.: Parallel algorithms for minimum cuts and maximum flows in planar networks. *J. ACM* 34, 950–967 (1987)

14 Johnson, D.B., Venkatesan, S.M.: Partition of Planar Flow Networks. In: *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pp. 259–264 (1983)

15 Khuller, S., Naor, J.: Flow in planar graphs with vertex capacities. *Algorithmica 11*, 200–225 (1994)

16 Kaplan, H., Nussbaum, Y.: Maximum flow in directed planar graphs with vertex capacities. *Algorithmica*, *in press*

17 Schmidt, F.R., Toppe, E., Cremers, D.: Efficient planar graph cuts with applications in Computer Vision. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 351–356 (2009)

18 Reif, J.H.: Minimum s-t cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM J. Comput.* 12, 71–81 (1983)