Minimum planar multi-sink cuts with connectivity priors*

Ivona Bezáková and Zachary Langley

Rochester Institute of Technology, Rochester, NY, U.S.A.
{ib,zbl9222}@cs.rit.edu

Abstract. Given is a connected positively weighted undirected planar graph G embedded in the plane, a source vertex s, and a set of sink vertices T. An (s,T)-cut in G corresponds to a cycle or a collection of edge-disjoint cycles in the planar dual graph G^* that define a planar region containing s but not T. A cut with a connectivity prior does not separate the vertices in T from each other: we focus on the most natural prior where the cut corresponds to a (simple, i. e., no repeated vertices) cycle in G^*. We present an algorithm that finds a minimum simple (s,T)-cut in $O(n^4)$ time for n vertices. To the best of our knowledge, this is the first polynomial-time algorithm for minimum cuts with connectivity priors. Such cuts have applications in computer vision and medical imaging.

1 Introduction

We address the problem of finding a minimum simple single-source-multi-sink cut in a positively weighted undirected planar graph $G = (V, E, w)$ embedded in the plane. In particular, given a source vertex s, and a set of sink vertices T, a cut $S \subseteq V$ is said to be *a simple* (s,T)*-cut* if S contains s and does not contain any vertex in T and the dual edges of the cut edges $\{(u, v) \mid u \in S, v \notin S\}$ form a simple cycle, i. e., no repeated vertices, in the dual graph. We present an $O(n^4)$ algorithm that finds a simple (s,T)-cut of the smallest weight in a positively weighted planar graph with n vertices. For a small example, see Figure 1(b).

Graph cuts are an important algorithmic tool in computer vision, see, e. g., [7, 6, 13]. For example, in the simplest form of image segmentation, a user is asked to identify a point (seed) inside an object and in the background. Viewing the input image as a graph, typically the 2d grid of pixels with edge weights representing (dis)similarity of neighboring pixels, a natural segmentation approach isolates the object by identifying a minimum cut between the two seeds. However, if the object contains thin parts (for example, if trying to isolate a vein on an ultrasound image), it is likely that a minimum cut will opt to sever the

* This material is based upon work supported by the National Science Foundation, Award No. CCF-1319987. Part of the work was done while the first author visited the Simons Institute for the Theory of Computing at the University of California, Berkeley.

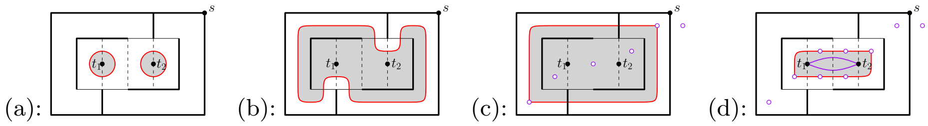

Fig. 1. (a) Minimum (s, T)-cut, weight 4ε: thick edges are of weight ∞, the five dashed edges of weight ε, and the other four edges of weight 1. (b) Minimum simple (s, T)-cut, weight $4 + 2\varepsilon$. (c)-(d) Common pitfalls involving shortest paths between sinks in the dual graph: (c) Merge all involved faces into a single face f and compute minimum (s, f)-cut; here results in weight 2∞. (d) Cut the plane along the shortest paths, find a shortest cycle that separates s from the "cut-out" face; here results in a non-simple cut (the middle two faces are visited twice). Small circles denote dual vertices.

thin parts from the object, see, e. g., [13]. A typical solution is to ask the user to identify multiple seeds inside the object. This might still result in the cut set containing several disconnected regions, as seen in the example in Figure 1(a). The problem can be remedied by enforcing a connectivity prior on the cut [13, 6, 14]. In other words, we do not look for a minimum cut but instead look for a smallest cut that somehow "connects" the seeds inside the object. Arguably the most natural connectivity prior is to require both the cut set as well as its complement to induce a connected graph. For connected planar inputs, this corresponds to finding a simple dual cycle that separates the seeds inside the object from the seed(s) outside. We also briefly discuss a connectivity prior where the cut corresponds to a non-self-crossing tour. While we admit that our running time is prohibitive for large inputs, we note that several applications first preprocess the input by contracting subcomponents, obtaining a much smaller graph. Our algorithms are to the best of our knowledge the first provably polynomial algorithms for minimum (s, T)-cut problems with connectivity priors.

The algorithms are based on a dynamic programming approach along a dual shortest path tree connecting the sinks. We first observe that there is a minimum simple (s, T)-cut such that its corresponding dual cycle does not cross this tree. However, the cycle might touch the tree arbitrarily many times, even along the same tree branch from both sides. In such case we form the cycle by concatenating paths that connect pairs of vertices on the tree. The tricky part is to ensure that the paths separate the source from the sinks and that the concatenation results in a simple cycle.

Instead of exactly computing the minimum length of such separating paths, we merely *bound* it to speed up the algorithm. In particular, we either get the shortest possible length, or the quantity we compute is smaller than the shortest length yet larger than the value of a minimum (s, T)-cut in which case the quantity will be eventually eliminated from the minimum cut computation. To guarantee that the cycle does not cross the tree, we "cut" the plane along the tree, thus preventing paths from crossing it. This involves duplicating each tree vertex as many times as is its degree in the tree, with no edges between the copies of the same vertex. The most challenging aspect of the computation is to ensure that we do not go through multiple copies of the same vertex – this is a problem with forbidden pairs of vertices, searching for a shortest cycle separating s and T, using at most one vertex of each forbidden pair. Our polynomial-time result contrasts with a related forbidden pair problem searching for a shortest cycle through a given vertex s, which is NP-hard even if the graph is planar and all forbidden pairs are on the outerface [8].

We remark that two natural, fast, and seemingly working approaches based on finding shortest sink-sink paths in the dual graph and contracting/merging components along the paths do not yield correct algorithms for this problem; see Figure 1(c)-(d).

We note that the single-source-multi-sinks problem in directed planar graphs remains open as our techniques are specific to undirected graphs since we expect to be able to traverse a path in both directions.

Related works. The case of a single sink t has been extensively studied as any minimum (s,t)-cut is simple. The latest algorithms run in time $O(n \log\log n)$ for undirected graphs due to Łącki and Sankowski [12], and in $O(n \log n)$ time for directed graphs, see, e. g., Borradaile and Klein [4] and the references within. Chalermsook, Fakcharoenphol, and Nanongkai [9] gave an $O(n \log^2 n)$ algorithm that finds an overall minimum cut (no pre-specified vertices to separate).

For multiple sinks, Chambers, de Verdière, Erickson, Lazarus, and Whittlesey [11] showed that the problem of finding a minimum simple multi-source-multi-sink cut in a planar graph is NP-hard (see the proof of Theorem 3.1). Bienstock and Monma [3] designed a polynomial-time algorithm that finds a shortest circuit separating a set of vertices from the outerface.

Several recent papers addresses various "multi" cut and flow problems in planar graphs. Bateni, Hajiaghayi, Klein, and Mathieu [1] designed a PTAS to approximate the weight of a minimum multiway cut where a set of terminals needs to be separated from each other. Borradaile, Klein, Mozes, Nussbaum, and Wulff-Nilsen [5] gave a near-linear algorithm for the multi-source-multi-sink flow problem.

For surfaces with genus g, Chambers, Erickson, and Nayyeri [10] designed $g^{O(g)} n \log n$ algorithms for finding minimum single-source-single-sink cuts in surface-embedded graphs. Chambers et al. [11] showed that the problem of finding a shortest splitting (i. e., simple and not homeomorphic to a disk) cycle is NP-hard but fixed parameter tractable with respect to g. Cabello [8] studied shortest contractible (i. e., homotopic to a constant function) and shortest separating (i. e., that split the surface into two connected components) cycles in embedded graphs, showing that a shortest contractible cycle can be found in polynomial time and the shortest separating cycle problem is NP-hard.

In a recent work [2], we investigated the existence and counting of "contiguous" cuts among all minimum single-source-multi-sink cuts in planar graphs. Unfortunately, the results heavily rely on the max-flow min-cut duality and do not extend to problems that go beyond minimum cuts.

2 Preliminaries

Let $G = (V, E, w)$ be a connected positively weighted undirected planar graph embedded in the plane. Its dual (multi-)graph $G^* = (V^*, E^*, w^*)$ is defined as follows. V^* is the set of faces of G. For every edge $e \in E$ bordering faces f_1, f_2, the set E^* contains the edge $e^* = (f_1, f_2)$ of the same weight as e, i. e., $w^*(e^*) = w(e)$. The planar embedding of G yields the corresponding planar embedding of G^*. For a given vertex $s \in V$ and a set of vertices $T \subseteq V$, $s \notin T$, an (s, T)-*cut* is a set of vertices $S \subseteq V$ such that $s \in S$ and $S \cap T = \emptyset$. The edges cut by the cut, i. e., $\{(u, v) \in E \mid u \in S, v \notin S\}$, correspond to dual edges $C = \{(u, v)^* \mid u \in S, v \notin S\}$. If C forms a simple cycle (no repeated vertices in the dual graph), then we say that the (s, T)-cut is *simple* or a *bond*. We allow C to be of length 1 (a dual self-loop), as well as of length 2 (formed by two different dual edges). The *weight* of an (s, T)-cut is the sum of the edge weights cut by the cut, i. e., $\sum_{(u,v)\in E: u\in S, v\notin S} w(u, v)$. For simple (s, T)-cuts this directly corresponds to the length of the corresponding dual cycle.

Observation 1 *A simple (s, T)-cut exists if and only if, after removing s from G, the sinks are still connected.*

Therefore, we assume that a simple (s, T)-cut exists. We also assume that $|T| > 1$ since for $|T| = 1$ any minimum (s, T)-cut is simple. Moreover, we assume that the degree of every vertex is > 1, as vertices of degree 1 can be merged with their neighbor. By a "cycle" or a "path" we mean a simple cycle or a simple path (no repeated vertices). If p is a path or a cycle, we denote by $|p|$ its length, i. e., the sum of the weights of edges on p. As we work with embedded graphs, we abuse terminology and identify a path or a cycle with the corresponding curve.

Let α and β be (possibly closed) non-self-intersecting curves in the plane. We say that α and β *cross* if there is a maximal curve χ (possibly just a point x) in $\alpha \cap \beta$ such that β continues on different sides of α at the end-points of χ; we refer to such χ as a cross segment. If α and β do not cross at χ, they *touch* at χ. If $\alpha \cap \beta \neq \emptyset$ and they share more than just their end-points, we say that α and β *intersect*.

Let G_0 be a graph obtained from G by enlarging each $u \in T \cup \{s\}$ into a small new face u^* with ∞-weighted edges bounding it. Notice that there is a straightforward bijection between simple (s, T)-cuts in G and simple cycles in $G_0^* - \{u^* \mid u \in T \cup \{s\}\}$ that define a planar region containing s^* but no $t^* \in T^*$, where $T^* := \{t^* \mid t \in T\}$. We refer to such cycles in G_0^* as (s^*, T^*)-*separating cycles*. To simplify our language, the remainder of this text takes place in the dual graph G_0^*. In particular, unless otherwise specified, by vertices and edges we mean vertices and edges of G_0^* and by the source and sinks we mean s^* and T^*.

Let c_{opt} be a minimum (s^*, T^*)-separating cycle in G_0^*. In this extended abstract we focus on the computation of the weight $|c_{opt}|$; the algorithm can be extended to obtain the corresponding cut within the same running time.

3 Graph H: cutting along a shortest path tree

Choose an arbitrary $t_1^* \in T^*$ and build a shortest[1] path tree τ from t_1^* to every $t_2^* \in T^*$, see Figure 2(a). Notice that every leaf of τ is a sink and that no sink nor s^* is an internal vertex of τ due to the ∞-weighted edges. Due to space constraints we omit the proof of the following lemma.

Lemma 1. *There exists a minimum (s^*, T^*)-separating cycle c such that c does not cross τ.*

Remark 1. We could have considered shortest paths between every pair of sinks which, in most cases, would have sped up the algorithm in practice. In the worst case, however, the union of all such paths forms a tree; using a tree simplifies the exposition in the paper.

Suppose we do a clockwise depth-first traversal of τ from t_1^* until we process the entire tree and return back to t_1^*. In a clockwise traversal, the neighbors of every vertex are considered in a clockwise cyclic manner (that is, if we use edge (u_1, v) to get to v, then we continue with edge (v, u_2) where u_2 is the clockwise next neighbor of v after u_1). Let $a_0, a_1, \ldots, a_{\ell-1}$ be the sequence of vertices encountered by the traversal, in this order and with repetitions – each vertex appears in the sequence as many times as is its degree in τ. See Figure 2(a).

We associate certain edges of G_0^* with each a_x: let $v_{x,0}, v_{x,1}, \ldots, v_{x,d_x+1}$ be the neighbors of a_x listed in the clockwise order from a_{x-1} to a_{x+1}, where $a_{-1} := a_{\ell-1}$ and $a_\ell := a_0$.

We construct a graph H analogous to G_0^* that will prevent us from crossing τ. Intuitively, the graph H corresponds to "cutting" the plane along the tree τ. We remove all vertices in τ and add a new vertex b_x for each a_x; we include edges of G_0^* that do not involve τ, plus edges (b_x, b_{x+1}) for each x, and edges from b_x to each $v_{x,i}$ for $1 \leq i \leq d_x$. (If $v_{x,i} = a_k$, then let $v_{x,i} := b_k$.) See Figure 2(b). Formally,

$$\begin{aligned}
V(H) &= V_0^* - V(\tau) \cup \{b_x \mid x \in \{0, 1, \ldots, \ell-1\}\}, \\
E(H) &= \{(u, v) \in E_0^* \mid u, v \in V_0^* - V(\tau)\} \cup \{(b_x, b_x + 1) \mid 0 \leq x < \ell\} \cup \\
&\quad \{e_{x,i} := (b_x, v_{x,i}) \mid 1 \leq i \leq d_x\}.
\end{aligned}$$

For convenience we define $e_{x,0} := (b_{x-1}, b_x)$ and $e_{x,d_x+1} := (b_x, b_{x+1})$. To simplify our expressions, we define $b_{x+y} := b_{(x+y) \bmod \ell}$ for $0 \leq x < \ell$ and $y \in \mathbf{Z}$ such that $x + y \notin \{0, 1, \ldots, \ell-1\}$. Also, let $B = \{b_0, \ldots, b_{\ell-1}\}$. Notice that every sink $t^* \in T^*$ corresponds to a unique a_{x_t} and therefore there is a unique corresponding vertex b_{x_t} in H. We abuse notation and use T^* and s^* to refer to the sinks and the source in H. Also, $b_0, \ldots, b_{\ell-1}$ is a cycle in H that bounds a single face f_B.

From now on the entire discussion takes place in H.

By *clockwise distance* from b_x to b_y, $x, y \in \{0, 1, \ldots, \ell-1\}$, we mean $y - x$ if $x \leq y$ and $\ell + y - x$ if $x > y$. Intuitively, this is the number of vertices,

[1] We allow the paths to use two ∞-weighted edges, at the start and at the end.

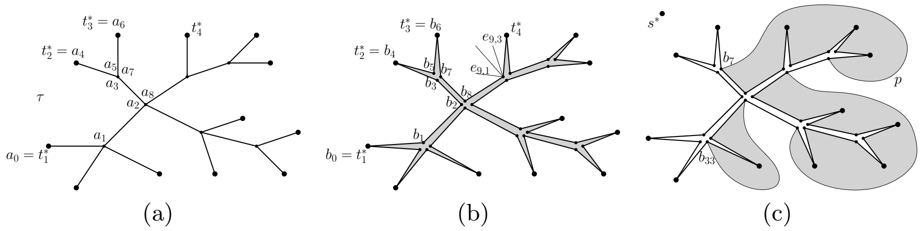

Fig. 2. (a) Tree τ, vertices a_x (notice that $a_2 = a_8$, $a_3 = a_5 = a_7$, etc.). (b) Graph H: vertices b_x and edges $e_{x,0}, \ldots, e_{x,d_x+1}$ (depicted are $e_{9,0} = (b_8, b_9)$, $e_{9,1}$, $e_{9,2}$, $e_{9,3}$, $e_{9,4} = (b_9, b_{10})$); face f_B is shaded. (c) Region R_p, here shown for a b_7-b_{33} source-sinks separating path p.

with repetition, we encounter during the traversal of τ from a_x to a_y. We write $b_x \prec b_y \prec b_z$ for $x, y, z \in \{0, 1, \ldots, \ell - 1\}$ if $x < y < z$, or, if $x > z$, then either $x < y$ or $y < z$. We say that such b_y is *clockwise between* b_x *and* b_z.

Lemma 2. *Let* $b_{i_1}, b_{i_2}, b_{i_3}, b_{i_4}$ *be such that* $b_{i_1} \prec b_{i_2} \prec b_{i_3} \prec b_{i_4} \prec b_{i_1}$. *Then,* $a_{i_1} = a_{i_3}$ *and* $a_{i_2} = a_{i_4}$ *cannot be both true.*

Proof. Suppose $a_{i_1} = a_{i_3}$ and suppose that as we traverse τ from a_{i_1}, we encounter a_{i_2} before returning back to $a_{i_1} = a_{i_3}$. Then, by the nature of the depth-first traversal, we must have processed all copies of a_{i_2} before returning back to a_{i_1}. Therefore, $a_{i_2} \neq a_{i_4}$. $\square$

Notice that any (s^*, T^*)-separating cycle in G_0^* that does not cross τ corresponds to a cycle in H that separates s^* from all T^*. The converse is not always true since a cycle in H may visit b_x, b_y where $x \neq y$ and hence $b_x \neq b_y$, yet $a_x = a_y$, leading to a non-simple cycle in G_0^*. We say that a cycle in H is (s^*, T^*)*-separating* if it yields a (simple) (s^*, T^*)-separating cycle in G_0^*.

4 Source-sinks separating paths

If a minimum (s^*, T^*)-separating cycle in G_0^* corresponds to a cycle in H that touches f_B, we will form it by concatenating paths between pairs of vertices in B. We need to be careful about concatenation of intersecting paths, as well as about separating s^* from T^*.

Definition 1. *Let p be a b_x-b_y path in H that does not go through the source or any of the sinks, and, if it goes through a vertex $b_z \in B$, then $b_x \preceq b_z \preceq b_y$. We define the* path-region(s) *R_p as the planar region(s) on the right of p, bounded by p and the clockwise b_x-b_y part of the boundary of f_B. We say that p is* source-sinks separating *if $s^* \notin R_p$ and that p is* no-B *if, except for b_x and b_y, it does not go through any vertices in B.*

Algorithm 1 Bounding the length of a shortest $e_{x,i}$-$e_{y,j}$ no-B source-sink separating path.

1: Let q be a shortest $e_{y,j}$-s^* path in H.
2: Construct H_q by "cutting" the plane open along q and removing vertices in $B \cup \{s^*\}$. In particular, for every vertex u on $q - \{b_y, s^*\}$, replace it by two new vertices u_1, u_2, and for every v such that $(u, v) \in E(H)$, add edge (u_1, v) if v is on the left of q, or (u_2, v) if it is on the right of q. Additionally, add edges (u_1, u_1') and (u_2, u_2') for every (u, u') on q and (v_1, v_2) for every $v_1, v_2 \in V(H) - B - q$ such that $(v_1, v_2) \in E(H)$. Use the same edge weights as in H.
3: For every $u \in q$, compute the shortest distance $\mathrm{dist}_{H_q}[v_{x,i}, u_1]$ from $v_{x,i}$ to u_1 where $e_{x,i} = (b_x, v_{x,i})$. Let $\mathrm{dist}_q[u, b_y]$ be the distance from u to b_y along q.
4: **return** $\beta[e_{x,i}, e_{y,j}] := w(e_{x,i}) + \min_{u \in q - \{b_y, s^*\}} \mathrm{dist}_{H_q}[v_{x,i}, u_1] + \mathrm{dist}_q[u, b_y]$.

The definition is depicted in Figure 2(c). Notice that all the sinks that lie clockwise between b_x and b_y are strictly inside $R_p \cup f_B$.

In this section we bound the length of a shortest b_x-b_y no-B source-sink separating path p that starts with the edge $e_{x,i}$ and ends with the edge $e_{y,j}$. We refer to such paths as $e_{x,i}$-$e_{y,j}$ no-B source-sink separating paths. Algorithm 1 summarizes the computation. It relies on the following lemmas; we omit their proofs due to space constraints. The proof of Lemma 4 is of a similar flavor as the forthcoming proof of Lemma 7. We note that the algorithm works even if $x = y$ (when searching for a no-B cycle c through b_x, the only vertex in B on c).

Lemma 3. *Let q be a shortest $e_{y,j}$-s^* path in $H - B \cup \{b_y\}$. There exists a shortest $e_{x,i}$-$e_{y,j}$ no-B source-sink separating path that does not cross q.*

Lemma 4. *Suppose there exists an $e_{x,i}$-$e_{y,j}$ no-B source-sink separating path in H; let p be a shortest such path. Then, the quantity computed by Algorithm 1 satisfies $\beta[e_{x,i}, e_{y,j}] \leq |p|$. Moreover, if $\beta[e_{x,i}, e_{y,j}]$ holds a numerical value, then $\beta[e_{x,i}, e_{y,j}] = |p|$, or $\beta[e_{x,i}, e_{y,j}] > |c_{opt}|$.*

Lemma 5. *Algorithm 1 can be implemented in time $O(n)$. Moreover, across different $e_{x,i}$, $e_{y,j}$ pairs, the computation of the length of the shortest $e_{x,i}$-$e_{y,j}$ no-B source-sink separating path can be done in overall time $O(n^2 \log n)$.*

5 Minimum (s^*, T^*)-separating cycle

If a minimum (s^*, T^*)-separating cycle goes through B, we decompose it into source-sinks separating paths of the following type.

Definition 2. *For $b_x, b_y \in B$, $b_x \neq b_y$, and i, j such that $0 \leq i \leq d_x$ and $1 < j \leq d_y + 1$, let $P[x, i, y, j]$ be the set of all b_x-b_y source-sinks-separating paths p in H such that*

1. *p leaves b_x by an edge $e_{x,i'}$ where $i < i'$,*
2. *p enters b_y by an edge $e_{y,j'}$ where $j' < j$, and*
3. *$a_{z_1} \neq a_{z_2}$ for every b_{z_1}, b_{z_2} on p where $b_{z_1} \neq b_{z_2}$.*

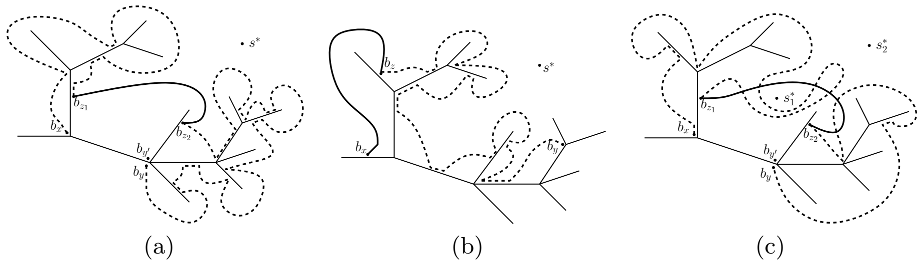

Fig. 3. (a)-(b): Possibilities for a shortest $e_{x,i}$-$e_{y,j}$ source-sinks separating path, schematic view: (a) Case 1: there exists $b_{y'}$ between b_x and b_y such that $a_{y'} = a_y$; the path is split into three subpaths p_1, p_2 (non-dashed, no-B), and p_3. (b) Case 3: there is no such $b_{y'}$ or $b_{x'}$. (c): Proof of Lemma 8, paths found by Algorithm 2 might intersect: Region R'_{s^*}. We show two possible locations for s^*: if $s^* = s_1^*$, then s^* is in a region bounded by subpaths of p_1', p_2', and p_3' and $L[x,i,y,j] > |c_{opt}|$; if $s^* = s_2^*$, we get a contradiction with the selection of p_1', p_2', and p_3'.

For $b_x = b_y$, $i < j$, let $P[x,i,y,j]$ be the set containing a single path, b_x, of length 0.

In other words, $p \in P[x,i,y,j]$ is a b_x-b_y source-sinks-separating path that leaves b_x by an edge that comes after $e_{x,i}$, enters b_y by an edge that comes before $e_{y,j}$, and it gives rise to a simple path in G_0^* (i. e., repeated vertices are not allowed). Recall also that a b_x-b_y source-sink separating path visits only those vertices in B that are clockwise between b_x and b_y.

We bound the length of a shortest path $p \in P[x,i,y,j]$ using dynamic programming. In particular, we compute $L[x,i,y,j]$ such that $L[x,i,y,j] = |p|$, or $|p| \geq L[x,i,y,j] > |c_{opt}|$. The algorithm, summarized in Algorithm 2, proceeds by gradually increasing the clockwise distance of b_x and b_y. Step 3 deals with the base case, $b_x = b_y$. For the inductive case, we distinguish three possibilities based on the relative position of b_x and b_y, as shown in Figure 3(a)-(b). Consider the $b_x, b_{x+1}, \ldots, b_y$ path in H and the corresponding walk $a_x, a_{x+1}, \ldots, a_y$ in G_0^*. Steps 7-9 (**case 1**) deal with the case when a_y is visited multiple times by the walk, steps 10-12 (**case 2**) with the case when a_x is visited multiple times, and steps 13-14 (**case 3**) with the case when both a_x and a_y are visited exactly once.

The following lemma analyzes the structure of a shortest path p in $P[x,i,y,j]$ and provides rationale for steps 7-9 of the algorithm; we defer analogous lemmas for steps 13-14 and 15 to the full version of the paper. The two subsequent lemmas provide bounds on $L[x,i,y,j]$.

Lemma 6. *Let p be a shortest path in $P[x,i,y,j]$. If b_x, b_y follow case 1, then there exist z_1, k_1, z_2, k_2 satisfying the conditions in step 9 such that p is a concatenation of paths p_1, p_2, and p_3, where $p_1 \in P[x,i,z_1,k_1]$, p_2 is an e_{z_1,k_1}-e_{z_2,k_2} no-B source-sinks-separating path, and $p_3 \in P[z_2,k_2,y,j]$.*

Proof. Since $a_y = a_{y'}$, path p needs to "jump" over $b_{y'}$ because of condition 3 of Definition 2. Let $b_{z_1} \in p$, $b_x \preceq b_{z_1} \prec b_{y'}$, be such that the clockwise distance from

Algorithm 2 Computing the weight of a minimum (s^*, T^*)-separating cycle.

Create graph H_b by contracting B into a single vertex b. Let L_0 be the weight of the minimum (s^*, b)-cut in H_b.
Compute $\beta[]$ for H using Algorithm 1.
Let $L[x, i, y, j] = 0$ if $x = y$ and $i < j$. Otherwise, $L[x, i, y, j]$ is undefined.
for d from 1 to $\ell - 1$ **do**
  for every x, y such that b_x and b_y are at clockwise distance d and $a_x \neq a_y$ **do**
    for every i, j, $0 \leq i \leq d_x + 1$, $0 \leq j \leq d_y + 1$ **do**
      if there is a $b_{y'}$, $b_x \prec b_{y'} \prec b_y$, such that $a_y = a_{y'}$ **then**
        Let $b_{y'}$ be such that $b_x \prec b_{y'} \prec b_y$, $a_y = a_{y'}$, and the clockwise distance from b_x to $b_{y'}$ is smallest possible.
        Let

$$L[x, i, y, j] := \min_{z_1, k_1, z_2, k_2} \{L[x, i, z_1, k_1] + \beta[e_{z_1,k_1}, e_{z_2,k_2}] + L[z_2, k_2, y, j]\},$$

        where z_1, k_1, z_2, k_2 range over all possibilities such that
        - $b_x \preceq b_{z_1} \prec b_{y'} \prec b_{z_2} \preceq b_y$,
        - $a_x \neq a_{z_1}$ or $b_x = b_{z_1}$, and $a_{z_2} \neq a_y$ or $b_{z_2} = b_y$, and
        - if $b_x = b_{z_1}$ then $i < k_1$, and if $b_{z_2} = b_y$ then $k_2 < j$.
      else
        if there is a $b_{x'}$, $b_x \prec b_{x'} \prec b_y$, such that $a_x = a_{x'}$ **then**
          Computation of $L[x, i, y, j]$ is analogous to the computation above.
        else
          Let

$$L[x, i, y, j] := \min_{i', z, k} \{\beta[e_{x,i'}, e_{z,k}] + L[z, k, y, j]\},$$

          where i', z, k range over all possibilities where $i < i'$ and $b_x \prec b_z \preceq b_y$.
return $L^* := \min\{L_0, \min_{x,i,y,j}\{\beta[e_{y,j}, e_{x,i}] + L[x, i, y, j]\}\}$, where x, i, y, j range over all possibilities such that either $b_x = b_y$ and $i < j$, or $a_x \neq a_y$, $i > 0$ and $j \leq d_y$.

b_{z_1} to $b_{y'}$ is smallest possible. Such z_1 must exist since p starts in B but it leaves it before it reaches $b_{y'}$. Let b_{z_2} be the next vertex in B, after b_{z_1}, encountered when traversing p from b_{z_1}. Note that b_{z_2} exists as p eventually gets to $b_y \in B$. Thus, p can be decomposed into several segments: a b_x-b_{z_1} path p_1, a b_{z_1}-b_{z_2} no-B path p_2, and a b_{z_2}-b_y path p_3.

Notice that $b_{y'} \prec b_{z_2} \preceq b_y$. This is because p enters vertices in B only between b_x and b_y and the p_1 segment blocks off access to B between b_x and b_{z_1}, and b_{z_1} is the clockwise closest vertex to $b_{y'}$ such that $b_{z_1} \in p$ and $b_x \preceq b_{z_1} \prec b_{y'}$. Also notice that for every $b_{z'}$ on p_3, it must be that $b_{z_2} \preceq b_{z'} \preceq b_y$, as access to vertices in B between b_x and b_{z_2} is blocked off by p_1 and p_2. Similarly, for every $b_{z'}$ on p_1, it must be that $b_x \preceq b_{z'} \preceq b_{z_1}$. Therefore, all the p_1, p_2, p_3 segments are source-sinks separating, since $s^* \notin R_p$.

Next we argue that $p_1 \in P[x, i, z_1, k_1]$, where k_1 is such that p leaves b_{z_1} by the edge e_{z_1,k_1}. If $b_x = b_{z_1}$, then, since $p \in P[x, i, y, j]$, condition 1 of Definition 2 implies that $i < k_1$. Therefore, we have $p_1 = b_x$ and $p_1 \in P[x, i, z_1, k_1]$. If $b_x \neq b_{z_1}$, then condition 1 of Definition 2 holds for $p_1 \in P[x, i, z_1, k_1]$ because

$p \in P[x,i,y,j]$ and p and p_1 share the starting edge. Condition 3 holds for p_1 since it holds for p. Condition 2 holds because if p_1 entered b_{z_1} by an edge $e_{z_1,k'}$, $k_1 \leq k'$, yet p leaves b_{z_1} by the edge e_{z_1,k_1} and then it continues to b_{z_2}, $b_x \preceq b_{z_1} \prec b_{z_2}$, we would get a loop, a contradiction with p being a path. Thus, $p_1 \in P[x,i,z_1,k_1]$.

By analogous reasons we have that $p_3 \in P[z_2,k_2,y,j]$, where k_2 is such that p enters b_{z_2} by the edge e_{z_2,k_2}. □

Lemma 7. *If $P[x,i,y,j] \neq \emptyset$, then $L[x,i,y,j] \leq |p|$, where p is a shortest path in $P[x,i,y,j]$.*

Proof. The proof proceeds by induction on the clockwise distance from b_x to b_y. The base case, $b_x = b_y$, follows from step 3 of Algorithm 2. For the inductive case, we distinguish the three cases for positions of b_x and b_y.

If b_x, b_y follow case 1, then p can be decomposed into p_1, p_2, and p_3 as described in Lemma 6. Let z_1, k_1, z_2, k_2 be the corresponding values. Then, $L[x,i,y,j] \leq L[x,i,z_1,k_1] + \beta[e_{z_1,k_1}, e_{z_2,k_2}] + L[z_2,k_2,y,j]$, since $L[]$ is computed as a minimization that considers z_1, k_1, z_2, k_2 as one of the options. By Lemma 4, $\beta[e_{z_1,k_1}, e_{z_2,k_2}] \leq |p_2|$. By the inductive hypothesis, $L[x,i,z_1,k_1] \leq |p'_1| \leq |p_1|$, where p'_1 is a shortest path in $P[x,i,z_1,k_1]$. Similarly, $L[z_2,k_2,y,j] \leq |p_2|$. Therefore, $L[x,i,y,j] \leq |p_1| + |p_2| + |p_3| = |p|$. Cases 2 and 3 are analogous. □

Lemma 8. *If $L[x,i,y,j]$ holds a numerical value, then $L[x,i,y,j] = |p|$, where p is a shortest path in $P[x,i,y,j]$, or $L[x,i,y,j] > |c_{opt}|$.*

Proof. We proceed by induction on the clockwise distance from b_x to b_y. For the base case, we have $b_x = b_y$, and step 3 computes $L[x,i,y,j]$ correctly.

For the inductive case, suppose that b_x and b_y fall under case 1. Let z'_1, k'_1, z'_2, k'_2 be the values that minimize the expression in step 9. Then, $L[x,i,y,j] = L[x,i,z'_1,k'_1] + \beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] + L[z'_2,k'_2,y,j]$. By the inductive hypothesis, $L[x,i,z'_1,k'_1] > |c_{opt}|$ or $L[x,i,z'_1,k'_1] = |p'_1|$, and $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] > |c_{opt}|$ or $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] = |p'_2|$, where p'_1 and p'_2 are shortest paths in $P[x,i,z'_1,k'_1]$ and $P[z'_2,k'_2,y,j]$, respectively. By Lemma 4, $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] > |c_{opt}|$ or $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] = |p'_2|$, where p'_2 is a shortest $e_{z'_1,k'_1}$-$e_{z'_2,k'_2}$ no-B source-sinks separating path. If $L[x,i,z'_1,k'_1] > |c_{opt}|$ or $L[z'_2,k'_2,y,j] > |c_{opt}|$ or $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] > |c_{opt}|$, we have $L[x,i,y,j] > |c_{opt}|$ since $L[x,i,z'_1,k_1]$, $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}]$, and $L[z'_2,k'_2,y,j]$ are nonnegative.

It remains to deal with the case when $L[x,i,z'_1,k'_1] = |p'_1|$, $\beta[e_{z'_1,k'_1}, e_{z'_2,k'_2}] = |p'_2|$, and $L[z'_2,k'_2,y,j] = |p'_3|$. By Lemma 7, we have $L[x,i,y,j] = |p'_1| + |p'_2| + |p'_3| \leq |p|$. Let p' be the concatenation of p'_1, p'_2, and p'_3. We will show that either p'_1, p'_2, and p'_3 do not intersect, in which case $p' \in P[x,i,y,j]$ and, therefore, $L[x,i,y,j] = |p'| = |p|$; or they do intersect, in which case $L[x,i,y,j] > |c_{opt}|$.

Claim. For every b_u, b_v, $b_u \neq b_v$, on p', we have $a_u \neq a_v$.

Proof: Suppose, by contradiction, that there are b_u, b_v, $b_x \preceq b_u \prec b_v \preceq b_y$, on p' such that $a_u = a_v$. Since $p'_1 \in P[x,i,z'_1,k'_1]$, it cannot be that both b_u and b_v

are on p'_1 due to condition 3 in Definition 2. Similarly, b_u and b_v cannot both be on p'_3. And, as p'_2 is a no-B path and $a_{z'_1} \neq a_{z'_2}$ due to Lemma 2 applied to z'_1, y', z'_2, y, vertices b_u and b_v cannot both be on p'_2. Recall also that p'_1 does not contain a vertex $b_{x'}$ with $a_{x'} = a_y$, as $b_{y'}$ has the smallest clockwise distance from b_x and it comes after $b_{z'_1}$. Thus, b_u is on p'_1 and b_v on p'_3. Since $a_u = a_v \neq a_y$, we get $b_u \prec b_{y'} \prec b_v \prec b_y$, a contradiction with Lemma 2. $\diamondsuit$

If p'_1, p'_2, and p'_3 do not intersect, then $p' \in P[x, i, y, j]$; this is due to the above claim and the p'_k's being from their respective $P[]$. Thus, $L[x, i, y, j] = |p'| = |p|$.

If p'_1, p'_2, and p'_3 intersect, their concatenation results in a walk with one or more loops, not a path. Let us look at the path-regions $R_{p'_1}$, $R_{p'_2}$, and $R_{p'_3}$. The union of these regions and f_B contains all the sinks between b_x and b_y strictly inside. Its complement contains the source. Let R' be the complement of $R_{p'_1} \cup R_{p'_2} \cup R_{p'_3} \cup f_B$. Let R'_{s^*} be the maximal simply connected planar region in R' that contains s^*. If R'_{s^*} borders no b_v, $b_y \preceq b_v \preceq b_x$, then R'_{s^*} is bounded by sub-paths of p'_1, p'_2 and p'_3, see Figure 3(c) where we assume $s^* = s^*_1$. Let c be the cycle in G^*_0 corresponding to the boundary of R'_{s^*}; notice that c is simple. Since R'_{s^*} does not contain any sinks, c is an (s^*, T^*)-separating cycle. Then, $L[x, i, y, j] = |p'_1| + |p'_2| + |p'_3| > |c| \geq |c_{opt}|$.

Finally, if R'_{s^*} borders some b_v for $b_y \preceq b_v \preceq b_x$, see Figure 3(c) (assume $s^* = s^*_2$), then R'_{s^*} is enclosed by the clockwise b_y-b_x part of the boundary of f_B and by a b_x-b_y path p'' that is formed by concatenating segments of p'_1, p'_2, and p'_3. Observe that $p'' \in P[x, i, y, j]$ and $|p''| < |p'_1| + |p'_2| + |p'_3| \leq |p|$. This is a contradiction with p being a shortest path in $P[x, i, y, j]$. $\square$

Corollary 1. *If $P[x, i, y, j] \neq \emptyset$, then $L[x, i, y, j] = |p|$ where p is a shortest path in $P[x, i, y, j]$, or $L[x, i, y, j] > |c_{opt}|$. If $P[x, i, y, j] = \emptyset$, then either $L[x, i, y, j]$ is undefined, or $L[x, i, y, j] > |c_{opt}|$.*

The corollary follows from observing that if $P[x, i, y, j] \neq \emptyset$, then $L[x, i, y, j]$ holds a numerical value. Now we are ready for the main theorem and the sketch of its proof; we defer the complete proof to the full version of the paper.

Theorem 1. *Algorithm 2 computes the weight of a minimum (s^*, T^*)-separating cycle in G^*_0 (and a minimum simple (s, T)-cut in G). It runs in time $O(n^4)$.*

Proof (sketch). Similarly as in Lemma 7, we get $L^* \leq |c_{opt}|$. Suppose $L^* < |c_{opt}|$. Since the value of L_0 corresponds to an (s^*, T^*)-separating cycle, we get $|c_{opt}| \leq L_0$. Thus, $L^* = \beta[e_{y',j'}, e_{x',i'}] + L[x', i', y', j']$ for some x', i', y', j'. If either quantity is $> |c_{opt}|$, we get $|c_{opt}| < |c_{opt}|$, a contradiction. If both quantities are computed correctly, we get a shorter (s^*, T^*)-separating cycle than c_{opt}, a contradiction. Therefore, $L^* = |c_{opt}|$.

The running time is $O(n^4)$ because there are $O(n^2)$ possibilities for $(x, i), (y, j)$ since they correspond to a pair of edges; and the computation $L[x, i, y, j]$ considers another $O(n^2)$ pairs of $(z_1, k_1), (z_2, k_2)$. $\square$

*Remark 2 (***Extensions).** The presented approach can be used for other connectivity priors. For example, a cut is contiguous if the dual cut-edges form a non-crossing tour that separates s^* from T^* (we allow repeated vertices but not edges as long as the tour can be drawn in a non-self-crossing manner in the infinitesimal neighborhood of each vertex). In a prior work [2], we computed how many of the minimum (s, T)-cuts are contiguous; however, the earlier approach did not extend to finding, among all contiguous cuts, the one with the smallest weight. Algorithm 2 can be modified to allow $a_x = a_y$, but one has to be careful not to use the edges of τ more than once. We leave the details for the journal version of this paper.

References

1. Bateni, M., Hajiaghayi, M., Klein, P.N., Mathieu, C.: A polynomial-time approximation scheme for planar multiway cut. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 639–655 (2012)
2. Bezáková, I., Langley, Z.: Contiguous minimum single-source-multi-sink cuts in weighted planar graphs. In: Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON). pp. 49–60 (2012)
3. Bienstock, D., Monma, C.L.: On the complexity of embedding planar graphs to minimize certain distance measures. Algorithmica 5(1), 93–109 (1990)
4. Borradaile, G., Klein, P.N.: An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. J. ACM 56(2) (2009)
5. Borradaile, G., Klein, P.N., Mozes, S., Nussbaum, Y., Wulff-Nilsen, C.: Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In: Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS). pp. 170–179 (2011)
6. Boykov, Y., Veksler, O.: Graph cuts in vision and graphics: Theories and applications (2006), in Handbook of Mathematical Models in Computer Vision (Springer)
7. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23(11), 1222–1239 (2001)
8. Cabello, S.: Finding shortest contractible and shortest separating cycles in embedded graphs. ACM Trans. on Algorithms 6(2) (2010), ext. abstr. in SODA '09.
9. Chalermsook, P., Fakcharoenphol, J., Nanongkai, D.: A deterministic near-linear time algorithm for finding minimum cuts in planar graphs. In: Proceedings of the 15th Annual ACM-SIAM Symp. on Discr. Algorithms (SODA). pp. 828–829 (2004)
10. Chambers, E.W., Erickson, J., Nayyeri, A.: Minimum cuts and shortest homologous cycles. In: Proceedings of the 25th Annual ACM Symposium on Computational Geometry (SCG). pp. 377–385 (2009)
11. Chambers, E.W., de Verdière, É.C., Erickson, J., Lazarus, F., Whittlesey, K.: Splitting (complicated) surfaces is hard. Comput. Geom. 41(1-2), 94–110 (2008)
12. Lacki, J., Sankowski, P.: Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In: Proceedings of the 19th Annual European Symposium on Algorithms (ESA). pp. 155–166 (2011)
13. Vicente, S., Kolmogorov, V., Rother, C.: Graph cut based image segmentation with connectivity priors. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) (2008)
14. Zeng, Y., Samaras, D., Chen, W., Peng, Q.: Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in N-D images. Computer Vision Image Understanding 112, 81–90 (2008)