Structured Recursive Separator Decompositions for Planar Graphs in Linear Time

[Extended Abstract] *

Philip N. Klein
Brown University
Providence, RI
klein@brown.edu

Shay Mozes
MIT
Cambridge, MA
shaym@mit.edu

Christian Sommer
csom@csail.mit.edu

ABSTRACT

Given a triangulated planar graph G on n vertices and an integer $r < n$, an *r–division* of G with *few holes* is a decomposition of G into $O(n/r)$ regions of size at most r such that each region contains at most a constant number of faces that are not faces of G (also called *holes*), and such that, for each region, the total number of vertices on these faces is $O(\sqrt{r})$.

We provide an algorithm for computing r–divisions with few holes in linear time. In fact, our algorithm computes a structure, called *decomposition tree*, which represents a recursive decomposition of G that includes r–divisions for essentially all values of r. In particular, given an exponentially increasing sequence $\vec{r} = (r_1, r_2, ...)$, our algorithm can produce a *recursive* $\vec{r}$–division with few holes in linear time.

r–divisions with few holes have been used in efficient algorithms to compute shortest paths, minimum cuts, and maximum flows. Our linear-time algorithm improves upon the decomposition algorithm used in the state-of-the-art algorithm for minimum *st*–cut (Italiano, Nussbaum, Sankowski, and Wulff-Nilsen, STOC 2011), removing one of the bottlenecks in the overall running time of their algorithm (analogously for minimum cut in planar and bounded-genus graphs).

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph Algorithms

Keywords

planar graphs; cycle separator; graph decomposition

1. INTRODUCTION

Separators decompose a graph in a *balanced* way into two subgraphs with a limited number of vertices in common. Separators are often used in efficient algorithms using a divide-and-conquer strategy [36, 10, 45, 44]. Graphs with small recursive separators include planar graphs [51, 35, 11, 38, 20, 48, 13], bounded-genus graphs [12, 21, 31], minor-free graphs [3, 2, 41, 42, 7, 30, 54], and graphs with bounded tree-width [23, 43, 6]. Furthermore, for graphs of all these classes, separators can be found efficiently, often in linear time. For planar graphs, experimental results demonstrate that separator algorithms are practical [1, 26, 18].

Perhaps the most influential result of this kind is the linear-time algorithm of Lipton and Tarjan [35] for finding a separator of size $O(\sqrt{n})$ in a planar graph with n vertices. Consider the result of using this algorithm recursively until each separated subgraph has size at most some specified limit r. It is easy to show that $O(n/r)$ subgraphs result, and that the average number of boundary vertices per subgraph is $O(\sqrt{r})$. Frederickson [19] showed that, with additional care, one can ensure that each subgraph has $O(\sqrt{r})$ boundary vertices; he named such a decomposition an *r–division*, and he referred to the subgraphs as *regions*. The running time of Frederickson's algorithm is $O(n \log r + (n/\sqrt{r}) \log n)$.

Decompositions of such kind have been used in many efficient algorithms for planar graphs, e.g. for computing shortest paths [19, 25], maximum flow [28], polygon triangulation [22], and I/O–efficient algorithms [37].

For some of these applications, Frederickson's algorithm for r–divisions is too slow. Goodrich [22] gave a linear-time algorithm that achieves $O(\sqrt{r})$ boundary size for planar graphs (see [4] for the I/O model). Even this was not enough for the linear-time shortest-path algorithm of Henzinger, Klein, Rao, and Subramanian [25], which requires recursive applications of an algorithm for r–divisions (with roughly $\log^* n$ levels of recursion). They addressed this by showing that a linear-time $O(\sqrt{n})$ separator algorithm could be used to obtain a sublinear-time separator algorithm with a worse (but still sublinear) boundary-size guarantee.

However, for some of the more involved algorithms working with planar embedded graphs, it is essential that the regions of the division are topologically "nice" in that the boundary of each region consists of a constant number of faces, also called *holes*. Such a division can be found by using small *cycle* separators (Miller [38]) instead of just small separators, and incorporating iterations in which the graph is separated according to the number of holes.

Such r–divisions with a constant number of holes were first used in algorithms of Klein and Subramanian [32, 49],

*A full version of this paper is available online at http://arxiv.org/abs/1208.2223

STOC'13, June 1–4, 2013, Palo Alto, California, USA.

and subsequently in many other algorithms [17, 9, 8, 40, 27, 34, 29, 16, 39, 33].[1]

Up to now, the fastest known algorithm computing an r–division with a constant number of holes per region runs in time $O(n \log r + (n/\sqrt{r}) \log n)$ [27]. This makes it one of the time bottlenecks in the state-of-the-art algorithms for minimum st–cut and maximum st–flow [27] and minimum cut [34] in undirected planar graphs and bounded-genus graphs [16]. Whether such an r–division can be computed in linear time was an open problem until the current work. For example, Cabello [8] remarks that *"it is unclear if the algorithm of Goodrich [22] can also be modified to use the cycle-separator, and thus obtain a linear-time construction of r–decompositions with a few holes."*

Contributions

We provide a linear-time algorithm for computing r–divisions with few holes for any triangulated biconnected plane graph G and any r (Theorem 2).

In fact, the algorithm produces a *decomposition tree* of G (Theorem 3), which is a tree that naturally represents a recursive decomposition of G by cycle separators, and from which one can read off, in linear time, a recursive $\vec{r}$–division with few holes, for any exponentially increasing sequence $\vec{r} = r_1, r_2, \ldots$ (Theorem 4).

Our linear-time algorithm improves upon the $O(n \log r + (n/\sqrt{r}) \log n)$–time algorithm in [27], removing one of the time bottlenecks of the state-of-the-art algorithms for minimum st–cut and maximum st–flow [27], as well as minimum cut [34, 16] (all in undirected graphs).

Concurrently with and independently of our result, Arge, van Walderveen, and Zeh [5] also gave an algorithm that produces an r–division with few holes in linear time. The focus of their paper is I/O-efficient algorithms. Their approach differs from ours in that their algorithm requires the value of r as input, and uses it in a way that makes it unsuitable for producing recursive $\vec{r}$–divisions in linear time. Such a linear time algorithm seems to be crucial for obtaining linear-time algorithms using r–divisions (see [25]).

Techniques

The overall approach of our algorithm builds on that of Goodrich [22], which is based on Lipton-Tarjan vertex separators [35]. However, our approach must handle the additional complexity of finding cycle separators (see Miller [38]), which involves maintaining spanning trees in both the input graph and its planar dual, and of bounding the number of holes. We manage both primal and dual trees simultaneously by using a primal spanning tree that follows a dual breadth-first-search tree (Section 4.2). The levels of the dual BFS tree define connected components, which we maintain in a *Component Tree* (Section 4.1). This component tree internally captures the structural connectivity of dual BFS components.

Outline

Precise definitions of r–divisions, holes, recursive divisions, and decomposition trees, as well as formal statements of our results are in Section 2.4 and Section 2.5. We give a high-level description of the r–division algorithm and prove its correctness in Section 3. A key ingredient in our algorithm is a cycle separator algorithm (Section 4) that can be implemented in sublinear time by using auxiliary data structures (based on *dynamic trees* and *Euler-tour trees*). This is crucial for achieving *overall* linear time. In our algorithm, we initialize the data structures once, and then, in subsequent steps of the recursion, we reuse what has been computed already (Section 5), and we call an efficient dynamic-tree implementation of our cycle separator algorithm (Section 4.5).

Some of the proofs and figures have been omitted from this extended abstract due to page limitations. See the full version for details.

2. PRELIMINARIES

2.1 Graph Notation

Let $G = (V, E)$ be a simple graph. For a subset V' of V, we denote by $\delta_G(V')$ the set of edges uv of G such that $v \in V'$ and $u \notin V'$. We refer to $\delta_G(V')$ as a *cut* in G. If V' and $V - V'$ both induce connected subgraphs of G, we say it is a *simple cut* (also known as a *bond*). In this case, V' and $V - V'$ are the vertex-sets of the two connected components of $\delta_G(V')$, which shows that the edges of $\delta_G(V')$ uniquely determine the bipartition $\{V', V - V'\}$.

A graph is called *biconnected* iff any pair of vertices is connected by at least two vertex-disoint paths.

For a spanning tree T of G and an edge e of G not in T, the *fundamental cycle* of e with respect to T in G is the simple cycle consisting of e and the unique simple path in T between the endpoints of e.

2.2 Some Properties of Planar Graphs

We assume the reader is familiar with the definitions of planar graphs and combinatorial embeddings.[2]

FACT 1 (SPARSITY). *Let G be a simple planar graph. $|E(G)| \leq 3\,|V(G)| - 6$.*

When the exact constants are not important we write $O(|G|)$ to denote $O(|V(G)|) = O(|E(G)|)$.

FACT 2 (SIMPLE-CYCLE/SIMPLE-CUT DUALITY [53]). *A set of edges forms a simple cycle in a planar embedded graph G iff it forms a simple cut in the dual G^*.*

Since a simple cut in a graph uniquely determines a bipartition of the vertices, a simple cycle in a planar embedded graph G uniquely determines a bipartition of the faces.

Definition 1. (Encloses) Let C be a simple cycle in a connected planar embedded graph G. Then the edges of C form a simple cut $\delta_{G^*}(S)$ for some set S of vertices of G^*, i.e. faces of G. Thus C uniquely determines a bipartition $\{F_0, F_1\}$ of the faces of G. Let f_∞, f be faces of G. We say C *encloses* f with respect to f_∞ if exactly one of f, f_∞ is in S. For a vertex/edge x, we say C *encloses* x (with respect to f_∞) if it encloses some face incident to x (encloses *strictly* if in addition x is not part of C).

FACT 3 ([52]). *For any spanning tree T of G, the set of edges of G not in T form a spanning tree of G^*.*

For a spanning tree T of G, we typically use T^* to denote the spanning tree of G^* consisting of the edges not in T. We often refer to T^* as the *cotree* of T [14].

[1] Cabello [8] requires only that the *average* number of holes per region be small.

[2] For elaboration, see `http://planarity.org`.

2.3 Separators

Definition 2. For an assignment $W(\cdot)$ of nonnegative weights to faces, edges, and vertices of G, we say a simple cycle C is a *balanced separator* if the total weight of faces, edges, and vertices strictly enclosed by C and the total weight not enclosed are each at most 3/4 of the total weight.

(Traditionally balance involves a bound of 2/3. We use 3/4 because it simplifies the presentation.)

One can reduce the case of face/edge/vertex weight to the case of face weight. For each vertex or edge, remove its weight and add it to an incident face. A cycle separator that is balanced with respect to the resulting face-weight assignment is balanced with respect to the original weight assignment. We may therefore assume in cycle-separator algorithms that only the faces have weight.

Let G be a planar embedded graph with face weights. Suppose that no face has more than 1/4 of the total weight. Lipton and Tarjan [35] show that, if G is triangulated (every face has size at most 3) then for any spanning tree T of G, there is an edge not in T whose fundamental cycle with respect to T is a balanced separator. Goodrich [22] observed that such an edge can be found by looking for an edge-separator in the cotree T^* of T.

We modify this approach slightly: let T^* be a spanning tree of the planar dual G^* of G such that T^* has maximum degree 3. Let T be the cotree of T^*, so T is a spanning tree of G. Root T^* at an arbitrary vertex of degree one or two. Let v be a leafmost vertex of T^* such that the descendants of v (including v) comprise more than 3/4 of the weight. Let $\hat{e}$ be the edge of T^* connecting v to a child with greatest descendant weight.

LEMMA 1. *The fundamental cycle of $\hat{e}$ with respect to T is a balanced simple cycle separator.*

PROOF. Simple algebra shows each of the two trees of $T^* \setminus \{\hat{e}\}$ comprises between 1/4 and 3/4 of the total weight. One of these trees consists of the faces enclosed by C, and the other consists of the faces not enclosed by C, where C is the fundamental cycle of $\hat{e}$ with respect to T. □

Miller [38] proved the following theorem

THEOREM 1 (MILLER [38]). *For a planar triangulated biconnected graph G with weights summing to W such that the weight of each face, edge, and vertex is at most $2W/3$, there is a linear-time algorithm that finds a simple cycle C of length at most $2\sqrt{2\,|V(G)|}$ such that the total weight strictly enclosed is at most $2W/3$, and the total weight not enclosed is at most $2W/3$.*

In this paper, we do not use Miller's construction. We give another construction that, with the aid of some auxiliary data structures, can be carried out in sublinear time. For simplicity of presentation, we present a construction that achieves a balance of 3/4 instead of 2/3, and refrain from optimizing the constants that arise in our construction.

2.4 Divisions

Frederickson [19] introduced the notion of r–divisions. Let $\bar{G}$ be an n-vertex planar embedded graph.

Definition 3. A *region* R of $\bar{G}$ is an edge-induced subgraph of $\bar{G}$.

Definition 4. A *division* of $\bar{G}$ is a collection of regions such that each edge is in at least one region. A vertex is a *boundary vertex* of the division if it is in more than one region. A division is an r–division if there are $O(n/r)$ regions, and each region has at most r vertices and $O(\sqrt{r})$ boundary vertices.

Frederickson's definition did not address the number of holes since it was not relevant in his algorithms (and in some subsequent algorithms building on his). The following definition follows the lines of that of Cabello [8] but our terminology is slightly different.

Definition 5. A *natural* face of a region R is a face of R that is also a face of $\bar{G}$. A *hole* of R is a face of R that is not natural.

Definition 6. An *r–division with few holes* is an r–division in which any edge of two regions is on a hole of each of them, and every region has $O(1)$ holes.

This differs from Cabello's definition in that his requires only that the *average* number of holes per region be $O(1)$. We use a stronger requirement because some algorithms depend on it.

THEOREM 2. *For a constant s, there is a linear-time algorithm that, for any biconnected triangulated planar graph $\bar{G}$ and any $r \geq s$, outputs an r–division of $\bar{G}$ with few holes.*

In fact, our algorithm further guarantees that all regions in the r–division are connected. This property is desirable in some applications (e.g., [33])

2.5 Recursive Divisions and Decomp. Trees

Some algorithms, e.g. the shortest-path algorithm of Henzinger et al. [25], require that the graph be decomposed into regions which are in turn decomposed into regions, and so on. That algorithm requires roughly $\log^* n$ levels of decomposition, so it would take more than linear time to find all the different divisions independently. We describe a simple decomposition of a planar graph that allows one to obtain such recursive divisions in linear time.

Definition 7. A *decomposition tree* for $\bar{G}$ is a rooted tree in which each leaf is assigned a region of $\bar{G}$ such that each edge of $\bar{G}$ is represented in some region. For each node x of the decomposition tree, the *region R_x corresponding to x* is the subgraph of $\bar{G}$ that is the union of the regions assigned to descendants of v.

Definition 8. A decomposition tree $\mathcal{T}$ *admits* an r–division with few holes if there is a set S of nodes of $\mathcal{T}$ whose corresponding regions form an r–division of $\bar{G}$ with few holes.

THEOREM 3. *For a constant s, there is a linear-time algorithm that, for any biconnected triangulated planar embedded graph G, outputs a binary decomposition tree $\mathcal{T}$ for G that admits an r–division of G with few holes for every $r \geq s$.*

Definition 9. For an exponentially increasing sequence $\vec{r} = (r_1, r_2, \ldots)$ of numbers, a *recursive $\vec{r}$–division of G with few holes* is a decomposition tree for G in which, for $i = 1, 2, \ldots$, the nodes at height i correspond to regions that form an r_i–division of G with few holes.

A recursive $\vec{r}$–division is exactly the kind of structure that is useful for the linear-time shortest-paths algorithm [25], although that algorithm does not require one with few holes.

THEOREM 4. *There is a linear-time algorithm that, given a decomposition tree satisfying the condition of Theorem 3, and given an increasing sequence $\vec{r}$, returns a* recursive $\vec{r}$–division of G with few holes.

Note that Theorems 2 and 4 follow easily from Theorem 3.

3. COMPUTING A DECOMP. TREE

In this Section we give a high-level description of the algorithm of Theorem 3 for computing a decomposition tree, and prove its correctness. The input graph $\bar{G}$ is assumed to be biconnected and triangulated. It follows that, for every region R of $\bar{G}$, every natural face is a triangle.

The algorithm is as follows. Given the input graph $\bar{G}$, the algorithm performs some preprocessing necessary for the sublinear-time cycle-separator algorithm (Algorithm 2), and calls RECURSIVEDIVIDE$(\bar{G}, 0)$. This main procedure is given in Algorithm 1. The parameter s is a constant appearing in the statement of Theorems 2 and 3.

Algorithm 1: RECURSIVEDIVIDE(R, ℓ)

```
1  let n = |V(R)|
2  if n ≤ s then return a decomposition tree consisting
   of a leaf assigned R
3  if ℓ mod 3 = 0 then separator chosen below to balance
   number of vertices
4  else if ℓ mod 3 = 1 then separator chosen to balance
   number of boundary vertices
5  else if ℓ mod 3 = 2 then separator chosen to balance
   number of holes
6  let R' be the graph obtained from R as follows:
   triangulate each hole by placing an artificial vertex in
   the face and connecting it via artificial edges to all
   occurrences of vertices on the boundary of the hole
7  find a balanced simple-cycle separator C in R' with at
   most c√n natural vertices
8  let F_0, F_1 be the sets of natural faces of R enclosed and
   not enclosed by C, respectively
9  for i ∈ {0, 1} do
10 |  let R_i be the region consisting of the edges of faces
   |  in F_i and edges of C that are in R
11 |  T_i ← RECURSIVEDIVIDE(R_i, ℓ + 1)
12 end
13 return the decomposition tree T consisting of a root
   with left subtree T_0 and right subtree T_1
```

This procedure, given a connected region R with more than s edges and given a recursion-depth parameter ℓ, first triangulates each hole of R by adding an artificial vertex and attaching it via artificial edges to each occurrence of a vertex on the boundary of the hole. Let R' be the resulting graph. See Figures 1(a), 1(b) for an illustration. The vertices and edges that are not artificial are *natural*. Triangulating in this way establishes biconnectivity of R'.

LEMMA 2. *R' is biconnected.*

Next, the procedure uses the SIMPLECYCLESEPARATOR procedure (Section 4.4) to find a simple-cycle separator C consisting of at most $c\sqrt{n}$ natural vertices, where c is a constant and $n := |V(R)|$ is the number of vertices of R (which is equivalent to the number of natural vertices of R'). Depending on the current recursion depth ℓ, the cycle separates R in a balanced way with respect to either vertices, boundary vertices, or holes.

Note that SIMPLECYCLESEPARATOR is called on R', which has more than n vertices since it also has some artificial vertices. However, we show in Lemma 5 that there are at most twelve artificial vertices, so even if we use a generic algorithm for finding a simple cycle separator, the bound of $c\sqrt{n}$ still holds for some choice of c (since $n \geq s$). In fact, our procedure SIMPLECYCLESEPARATOR takes into account which vertices are artificial, and returns a separator consisting of at most $4\sqrt{3n}$ natural vertices, and possibly also some artificial vertices. The number of artificial vertices on the separator does not matter in the analysis of RECURSIVEDIVIDE.

The cycle C determines a bipartition of the faces of the triangulated graph R', which in turn induces a bipartition (F_0, F_1) of the natural faces of R. For $i \in \{0, 1\}$, let R_i be the region consisting of the edges bounding the faces in F_i, together with the edges of C that are in R (i.e. omitting the artificial edges added to triangulate the artificial faces). See Figures 1(c), 1(d) for an illustration.

LEMMA 3. *If R is connected then R_0 is connected and R_1 is connected.*

The procedure calls itself recursively on R_0 and R_1, obtaining decomposition trees $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively. The procedure creates a new decomposition tree $\mathcal{T}$ by creating a new root corresponding to the region R and assigning as its children the roots of $\mathcal{T}_0$ and $\mathcal{T}_1$.

3.1 Number of Holes

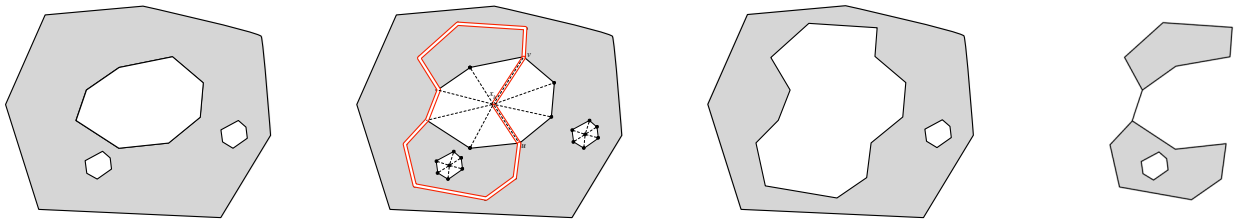
The triangulation step (Line 6) divides each hole h into a collection of triangle faces. We say a hole h is *fully enclosed by C* if all these triangle faces are enclosed by C in R'.

LEMMA 4. *Suppose that there are k holes that are fully enclosed by C. Then R_0 has $k + 1$ holes.*

PROOF. We give an algorithmic proof. See Figure 1 for an illustration. Initialize R'_0 to be the graph obtained from R' by deleting all edges not enclosed by C. Then C is the boundary of the infinite face of R'_0. Consider in turn each hole h of R such that a nonempty proper subset of h's triangle faces are enclosed by C. For each such face h, C includes the artificial vertex x_h placed in h, along with two incident edges ux_h and x_hv where u and v are distinct vertices on the boundary of h. Deleting all the remaining artificial edges of h modifies the boundary of the infinite face by replacing $ux_h\ x_hv$ with a subsequence of the edges forming the boundary of h. In particular, deleting these artificial edges does not create any new faces.

Finally, for each hole h that is fully enclosed by C, delete the artificial edges of h, turning h into a face of R'_0. The resulting graph is R_0, whose holes are: the holes of R that were fully enclosed by C, together with the infinite face of R_0. □

If the recursion depth mod 3 is 2, Line 7 of RECURSIVEDIVIDE must select a simple cycle in R' that is balanced with respect to the number of holes. To achieve this, for each hole h of R, the algorithm assigns weight 1 to one of the triangles resulting from triangulating h in Line 6, and weight 0



(a) (b) (c) (d)

Figure 1: Illustration of triangulating a hole and separating along a cycle. 1(a) A schematic diagram of a region R with four holes (white faces, one of them being the unbounded face). 1(b) The graph R' and a cycle separator C (double-lined red). Artificial triangulation edges are dashed (triangulation edges are not shown for the unbounded hole to avoid clutter). 1(c) The region R_0 consisting of the edges bounding the faces not enclosed by C together with the edges of C that belong to R. Equivalently, R_0 is the subgraph of R' not strictly enclosed by C without any artificial edges and vertices. R_0 has three holes. 1(d) The region R_1 consisting of the edges bounding the faces enclosed by C together with the edges of C that belong to R. Equivalently, R_1 is the subgraph of R' enclosed by C without any artificial edges and vertices. R_1 has two holes. Note that a hole is not necessarily a simple face.

to all other faces. Then the algorithm finds a cycle C that is balanced with respect to these face-weights.

The following lemma, whose proof follows from Lemma 4, establishes a constant bound on the number of holes.

LEMMA 5. *For any region created by* RECURSIVEDIVIDE*, the number of holes is at most twelve.*

3.2 Number of Vertices and Boundary Vertices

If the recursion depth mod 3 is 0, Line 7 of RECURSIVEDIVIDE selects a simple cycle in R' that is balanced with respect to the number of natural vertices. To achieve this, for each natural vertex v, the algorithm selects an adjacent face in R', dedicated to carry v's weight. The weight of each face is defined to be the number of vertices for which that face was selected. Since each face in R' is a triangle, every weight is an integer between 0 and 3. A cycle C is then chosen that is balanced with respect to these face-weights.

If the recursion depth mod 3 is 1, the cycle must be balanced with respect to the number of boundary vertices. For each boundary vertex, the algorithm selects an incident face; the algorithm then proceeds as above.

In either case, the total weight enclosed by the cycle C is an upper bound on the number of vertices (natural or boundary) strictly enclosed by C. Thus at most three-fourths of the vertices (natural or boundary) of R' are strictly enclosed by C in R'. Similarly, at most three-fourths of the vertices are not enclosed by C in R'.

The vertices of R_0 are the natural vertices of R' enclosed by C (including the natural vertices on C, which number at most $c\sqrt{|V(R)|}$), and the vertices of R_1 are the natural vertices of R' not strictly enclosed by C. Let $n = |V(R)|$ and, for $i \in \{0, 1\}$, let $n_i = |V(R_i)|$. We obtain

$$n_0 + n_1 \leq n + c\sqrt{n}. \tag{1}$$

Moreover, if the recursion depth mod 3 is 0, then

$$\max\{n_0, n_1\} \leq \frac{3}{4}n + c\sqrt{n}. \tag{2}$$

Similarly, let b be the number of boundary vertices of R, and, for $i \in \{0, 1\}$, let b_i be the number of boundary vertices of R_i. We obtain

$$b_0 + b_1 \leq b + c\sqrt{n}. \tag{3}$$

Moreover, if the recursion depth mod 3 is 1, then

$$\max\{b_0, b_1\} \leq \frac{3}{4}b + c\sqrt{n}. \tag{4}$$

3.3 Admitting an r–division

Let N be the number of vertices in the original input graph $\bar{G}$. Consider the decomposition tree $\mathcal{T}$ of $\bar{G}$ produced by RECURSIVEDIVIDE. Each node x corresponds to a region R_x. We define $n(x) = |V(R_x)|$. In this section we show that, for any given $r \geq s$, $\mathcal{T}$ admits an r–division of $\bar{G}$. We adapt the two-phase analysis of Frederickson [19]. In the first phase (Lemma 6), we identify a set of $O(N/r)$ regions for which the average number of boundary vertices is $O(\sqrt{r})$. However, some of the individual regions in this set might have too many boundary vertices (since the number of vertices and boundary vertices do not necessarily decrease at the same rate). We show that each such region can be replaced with smaller regions in $\mathcal{T}$ so that every region has $O(\sqrt{r})$ boundary vertices, and the total number of regions remains $O(N/r)$ (Lemma 7).

For a node x of $\mathcal{T}$ and a set S of descendants of x such that no node in S is an ancestor of any other, define $L(x, S) = -n(x) + \sum_{y \in S} n(y)$. Roughly speaking, $L(x, S)$ counts the number of new boundary nodes with multiplicities when replacing x by all regions in S. If the children of x are x_0 and x_1, Equation (1) implies

$$L(x, \{x_0, x_1\}) \leq c\sqrt{n(x)}. \tag{5}$$

Fix r and let S_r be the set of nodes y of $\mathcal{T}$ such that y's region has no more than r vertices but the region of y's parent has more than r vertices. Note that no node in S_r is an ancestor of any other. Let $\hat{x}$ be the root of $\mathcal{T}$.

LEMMA 6 (TOTAL NUMBER OF BOUNDARY VERTICES). *There are constants s and γ, depending on c, such that, for any $r > s$, $L(\hat{x}, S_r) \leq \frac{\gamma N}{\sqrt{r}}$.*

The proof of Lemma 6, which is included in the full version of this paper, is an adaptation of Frederickson's analysis.

Lemma 6 implies that $\sum_{x \in S_r} n(x)$ is $O(N)$, since the regions in S_r are disjoint *except* for boundary vertices, of which there are at most $O(N/\sqrt{r})$. For each parent y of a node $x \in S_r$, the corresponding region R_y has more than r vertices, so the number of such parents is $O(N/r)$, so $|S_r|$ is $O(N/r)$. Let c' be a constant to be determined. For a node x, let $S'_r(x)$ denote the set of rootmost descendants y of x (where x is a descendant of itself) such that R_y has at most $c'\sqrt{r}$ boundary vertices. Let $S'_r = \bigcup_{x \in S_r} \{S'_r(x)\}$.

LEMMA 7. *The regions $\{R_y : y \in S'_r\}$ form an r–division with a constant number of holes per region.*

The proof is also an adaptation of Frederickson's argument.

4. A SIMPLE-CYCLE SEPARATOR ALGORITHM

In this section we present our cycle separator algorithm. As a one-shot algorithm, the input is a simple biconnected graph G with m edges and face weights such that no face weighs more than 3/4 the total weight and no face consists of more than 3 edges. The algorithm outputs a simple cycle C in G, such that neither the total weight strictly enclosed by C nor the total weight not enclosed by C exceeds 3/4 of the total weight. The length of C is guaranteed to be at most $4\sqrt{|E(G)|}$. Since G is a simple graph, this implies a bound of $4\sqrt{3\,|V(G)|}$ on the length of the cycle. Better constants can be achieved, at the cost of complicating the algorithm and the analysis, which we avoid for the sake of simplicity and ease of presentation. In a similar manner, we aim for 3/4–balance to simplify the presentation. A balance of 2/3 can be achieved.

The cycle separator algorithm consists of a preprocessing step, which runs in linear time and computes certain auxiliary data structures used by the main procedure, SIMPLECYCLESEPARATOR, which performs the computation. These data structures can be represented so that SIMPLECYCLESEPARATOR takes sublinear time. One auxiliary data structure is a tree $\mathcal{K}$, which is called the *component tree*. The tree $\mathcal{K}$ captures the structural connectivity of dual BFS components of G. The dual BFS components satisfy a certain disjointness property (see Lemma 9). The other auxiliary data structures are a spanning tree T of G and its cotree T^*. The spanning tree T satisfies a certain monotonicity property (see Lemma 10). The disjointness and monotonicity properties guarantee that the length of the cycle separator output by SIMPLECYCLESEPARATOR is $4\sqrt{|E(G)|}$.

Our algorithm for constructing the decomposition tree $\mathcal{T}$ invokes SIMPLECYCLESEPARATOR multiple times, on various regions of $\bar{G}$. It computes the auxiliary data structures $\mathcal{K}$, T, and T^* once, for the input graph $\bar{G}$, and efficiently updates their representation when separating a region into two regions. However, the disjointness and monotonicity properties mentioned above are slightly weaker (see Invariant 1 and Invariant 2); they only apply to natural edges (disjointness) and to natural vertices (monotonicity). One implication is that the number of natural vertices on the cycle separator C produced by SIMPLECYCLESEPARATOR is bounded in terms of the (square root of the) number of natural vertices in the input graph. However, the cycle C might also include some artificial vertices. As argued in Section 3, artificial nodes on C do not affect the analysis.

4.1 Levels and Level Components

We define levels with respect to an arbitrarily chosen face f_∞, which we designate as the infinite face.

Definition 10. The *level* $\ell^F(f)$ of a face f is the minimum number of edges on a f_∞–to–f–path in G^*. We use L_i^F to denote the faces having level i, and we use $L_{\geqslant i}^F$ denote the set of faces f having level at least i.

Definition 11. For an integer $i \geq 0$, a connected component of the subgraph of G^* induced by $L_{\geqslant i}^F$ is called a *level-i component*, or, for unspecified i, a *level component*. We use $\mathcal{K}_{\geqslant i}$ to denote the set of level-i components. A level-i component K is said to have level i, and we denote its level by $\ell^{\mathcal{K}}(K)$. A *non-root* level component is a level component whose level is not zero. The set of vertices of G^* (faces of G) belonging to K is denoted $F(K)$.

Note that we use K (not K^*) to denote a level component even though it is a connected component of a subgraph of the planar dual. K should be thought of as a set of faces. Thus we can refer to it as a subgraph of G^* or of G. In the former case K is the subgraph of G^* induced by the faces in the set K. In the latter case K is the subgraph of G induced by the edges that belong to faces in the set K.

LEMMA 8. *For any non-root level component, the subgraph of G^* consisting of vertices of G^* not in $F(K)$ is connected.*

COROLLARY 1. *For any non-root level component K, the edges of $\delta_{G^*}(F(K))$ form a simple cycle in the primal G.*

In view of Corollary 1, for any non-root level component K, we use $X(K)$ to denote the simple cycle in the primal G consisting of the edges of $\delta_{G^*}(F(K))$. We refer to $X(K)$ as the *bounding cycle* of K since, when viewed as a subgraph of G, K is exactly the subgraph enclosed by $X(K)$.

LEMMA 9. *Let K and K' be two distinct components. $X(K)$ and $X(K')$ are edge-disjoint.*

PROOF. Let i be the level of K. The edges of $X(K)$ are edges of $\delta_{G^*}(F(K))$. Therefore, as edges of G^*, they have one endpoint in K and one in a level-$(i-1)$ face. If an edge of $\delta_{G^*}(F(K'))$ has an endpoint in K then $K \neq K'$ implies the level of K' is at least $i+1$, so it cannot be an edge of $X(K)$. If, on the other hand, an edge of $\delta_{G^*}(F(K'))$ has an endpoint in level $i-1$ then $K \neq K'$ implies the other endpoint is not in K. □

Definition 12. The *component tree* $\mathcal{K}$ is the rooted tree whose nodes are the level components and in which K is an ancestor of K' if the faces of K include the faces of K'.

The root of the component tree is the unique level-0 component consisting of all of G^*.

Definition 13. An edge ff' of G^* has level i if f has level $(i-1)$ and f' has level i. We write $\ell^E(ff')$ for the level of ff'. We use L_i^E to denote the set of edges of level i.

Note that not every edge of G^* is assigned a level.

Definition 14. Let L_i^V denote the set of vertices of the primal graph G that are endpoints in the primal graph G of edges in L_i^E.

Note that a vertex of G can be an endpoint of two edges at different levels i and j, so L_i^V and L_j^V are not necessarily disjoint.

Definition 15. The *level* $\ell^V(v)$ of a primal vertex v is defined to be $\min_f \ell^F(f)$ over all faces f incident to v.

Note that L_i^V is *not* the set of vertices with level i.

4.2 The Primal Tree

The algorithm maintains a primal spanning tree T. We start by describing the initial value of T and its properties.

LEMMA 10. *There exists a spanning tree $\bar{T}$ such that, for any vertex u, $\ell^V(\text{parent}_{\bar{T}}(u)) < \ell^V(u)$. $\bar{T}$ can be computed in linear time.*

PROOF. For a primal vertex u with $\ell^V(u) = i$, let f be a level-i face to which u is incident (ties are broken arbitrarily, but consistently). Let v, w be the other two vertices of that face f. The parent of u in $\bar{T}$, denoted by $\text{parent}_{\bar{T}}(u)$, is the vertex in $\{v, w\}$ with the smaller level (again, breaking ties arbitrarily and consistently).

Since $\ell^V(u) = i$, u is not incident to a level-$(i-1)$ face. Hence both v and w must be adjacent to a level-$(i-1)$ face f'. Therefore, $\ell^V(\text{parent}_{\bar{T}}(u)) \le i - 1 < \ell^V(u) = i$.

To complete the definition of $\bar{T}$, we choose an arbitrary vertex r incident to f_∞ to be the root of $\bar{T}$ by assigning it to be the parent of the two remaining vertices at level $i = 0$. For convenience we set the level of the root vertex to -1. □

4.3 The Preprocessing Step

To compute a simple cycle separator, one first computes the component tree $\mathcal{K}$, the spanning tree T, and its cotree T^*. This is done by the preprocessing step (Algorithm 2), which runs in linear time.

Algorithm 2: PREPROCESSING(G)

```
1  choose an arbitrary face as f_∞
2  compute face, edge, and vertex levels ℓ^F(·), ℓ^E(·), ℓ^V(·), respectively
3  compute the component tree 𝒦
4  initialize T to be the tree T̄ as defined in Lemma 10
5  initialize T* to be the cotree of T
6  return (𝒦, T, T*)
```

The efficient implementation of RECURSIVEDIVIDE, which computes the decomposition tree $\mathcal{T}$ in linear time, recursively separates regions of $\bar{G}$. It maintains the component tree $\mathcal{K}$, spanning tree T, and cotree T^* of the currently handled region R throughout the recursive calls. This is described in Section 5. T is initialized to be the tree $\bar{T}$ of $\bar{G}$. RECURSIVEDIVIDE maintains the following invariants:

INVARIANT 1. *Let K and K' be two distinct components. $X(K)$ and $X(K')$ do not share natural edges.*

INVARIANT 2. *$\ell^V(v) < \ell^V(u)$ for any two natural vertices u and v of R such that v is a proper ancestor of u in T.*

Since $\bar{G}$ has only natural vertices and edges, Lemma 9 and Lemma 10 show that the invariants initially hold for $R = \bar{G}$.

Algorithm 3: SIMPLECYCLESEPARATOR($\mathcal{G}$), $\mathcal{G}$ is the tuple $(G, \mathcal{K}, T, T^*)$

```
1  let m be the number of natural edges in G
2  let W := W(G)
3  compute e* to be the 3/4–balanced edge separator of T*
4  let e be the primal of e*
5  let C̃ denote the fundamental cycle of e w.r.t. T
6  if C̃ has at most 4√m natural edges then return C̃
7  let l, h be the minimum and maximum level ℓ^V(v) of a vertex v ∈ C̃
8  while l < h do /* binary search for i_0 in [l, h) */
9      set all i_0, i_−, i_+ to l + ⌊(h − l)/2⌋
10     repeat /* sequential search for level i_− < i_0 with small boundary */
11         i_− := i_− − 1; let K_− be the unique component at level i_− that C̃ intersects
12     until i_0 < l or X(K_−) has at most √m natural edges
13     if W(G \ K_−) > 3W/4 then h := i_−; continue
14     repeat /* sequential search for level i_+ > i_0 with small boundary */
15         i_+ := i_+ + 1; let K_+ be the unique component of level i_+ that C̃ intersects
16     until i_+ > h or X(K_+) has at most √m natural edges
17     if W(K_+) > 3W/4 then l := i_+; continue
18     let G' be the graph induced on G by
           X(K_−) ∪ X(K_+) ∪ (C̃ ∩ (K_− − K_+))
19     return GREEDYCYCLESEPARATOR(G')
20 end
```

4.4 Computing a Simple Cycle Separator

We first provide an intuitive description of the separator algorithm SIMPLECYCLESEPARATOR (Algorithm 3). Let m denote the number of natural edges in G.

1. The algorithm computes a balanced fundamental cycle $\tilde{C}$ (Lines 3–5). If $\tilde{C}$ consists of fewer than $4\sqrt{m}$ natural edges, then $\tilde{C}$ is a short balanced simple-cycle separator.
2. Otherwise, the fact that $\tilde{C}$ is long implies that it intersects components at many (more than $2\sqrt{m}$) consecutive levels of the component tree $\mathcal{K}$. (we say that $\tilde{C}$ intersects a component K if $\tilde{C}$ has at least one edge in $K - X(K)$ and at least one in $G - K$). The algorithm performs a binary search procedure on the range $[l, h)$, where l, h are the minimum and maximum level $\ell^V(v)$ of a vertex $v \in \tilde{C}$, respectively. At each step of the binary search, we identify
   - the median level $i_0 = \lfloor (l + h)/2 \rfloor$,
   - the highest-level component K_- intersected by $\tilde{C}$, whose level i_- is smaller than i_0 and whose bounding cycle $X(K_-)$ has few (at most $\sqrt{m}$) natural edges, and
   - the lowest-level component K_+ intersected by $\tilde{C}$, whose level i_+ is at least i_0 and whose bounding cycle $X(K_+)$ has few natural edges.

The monotonicity of the primal tree T implies that the number of natural edges of $\tilde{C}$ between levels i_- and i_+ (that is, the number of edges of $\tilde{C}$ in $K_- - K_+$) is at most $2\sqrt{m}$ (see Lemma 13).

3. If $W(G - K_-) > 3W/4$ we continue the binary search on the range $[l, i_-)$. Similarly, if $W(K_+) > 3W/4$ we continue the binary search on the range $[i_+, h)$.

4. Otherwise, the graph G' induced by the edges of $X(K_-)$, $X(K_+)$, and the edges of $\tilde{C}$ in $K_- - K_+$ is a biconnected planar graph with at most $4\sqrt{m}$ natural edges, none of whose faces weighs more than $3W/4$. See Figure 2. The algorithm uses a greedy procedure, GreedyCycleSeparator, to output a balanced simple cycle separator in G' in $O(\sqrt{m})$ time.

The pseudocode of SimpleCycleSeparator is given in Algorithm 3. To avoid clutter, the pseudocode does not handle the boundary case where the loop in line 11 terminates without finding a level K_- with $|X(K_-)| < \sqrt{m}$. In this case there is no need to shortcut $\tilde{C}$ at a small level. Specifically, $\tilde{C}$ is entirely enclosed by K_-, the condition in line 13 is considered false, and $X(K_-)$ is considered to be an empty set of edges. A similar statement applies to K_+.

Lemma 11. *The components K_- and K_+ defined in Lines 11 and 15 are well defined.*

Proof. Since $l \le i_- \le h$, the cycle $\tilde{C}$ must intersect some component at level i_-. The monotonicity of the tree T implies that once a rootward path leaves a component at some level it never enters any component at that level again. The lemma follows since $\tilde{C}$ is comprised of two rootward paths plus one edge. □

Lemma 12. *For $l \le i \le h$, let K_i be the unique component at level i that is intersected by $\tilde{C}$. If the weight not enclosed by K_- is greater than $3W/4$ then there exists a level $l \le i_0 < i_-$ such that the weight not enclosed by any K_i with $l \le i \le i_0$ is at most $3W/4$, and such that the weight enclosed by any K_i with $i_0 < i \le h$ is at most $3W/4$.*

Proof. Let K_l be the unique level-l component that encloses $\tilde{C}$. Since $\tilde{C}$ is a balanced separator, the weight enclosed by K_l is at least $W/4$. Hence the weight not enclosed by K_l is at most $3W/4$. Let $\tilde{K}$ be the component with maximum level that is intersected by $\tilde{C}$ and whose enclosed weight is at least $W/4$. Let i_0 be the level of $\tilde{K}$. By the above argument $l \le i_0$, and since the weight not enclosed by K_- is greater than $3W/4$, $i_0 < i_-$. By choice of i_0, any K_i with $i_0 < i \le h$ encloses at most $W/4$ weight. □

A symmetric lemma applies to the case where the weight enclosed by K_+ is greater than $3W/4$. These lemmas show that, if l or h are updated (i.e., the binary search continues), there exists some level i_0 in the new search range for which both conditions in Lines 13 and 17 are false, and hence the binary search procedure must eventually terminate.

Lemma 13. *Let G be a biconnected plane graph G with m natural edges and face weights such that no face weighs more than $3/4$ the total weight and such that no face consists of more than 3 edges. The procedure* SimpleCycleSeparator *finds a $3/4$–balanced simple cycle separator in G with at most $4\sqrt{m}$ natural vertices.*

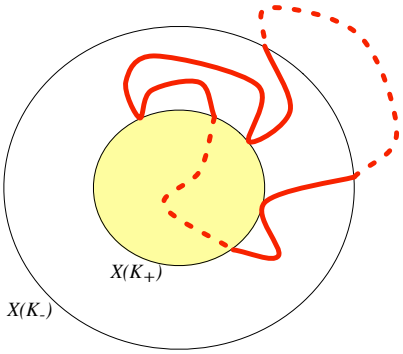


Figure 2: Illustration of the cycle separator algorithm: the fundamental cycle $\tilde{C}$ is shown in thick red (solid and dashed). The level cycles $X(K_-)$ and $X(K_+)$ are indicated. The component K_+ is shaded yellow. G' is the graph consisting of the edges of $\tilde{C}$ that belong to $K_- - K_+$ (solid thick red) as well as the edges of the cycles $X(K_-)$ and $X(K_+)$ (black).

Proof. Since G has face size at most 3, the fundamental cycle $\tilde{C}$ is a $3/4$–balanced simple cycle separator. If it consists of fewer than $4\sqrt{m}$ natural edges it is returned in Line 5. Otherwise, the lemma follows from the correctness of GreedyCycleSeparator (Lemma 14), provided that we show that G' is a biconnected subgraph of G with $4\sqrt{m}$ natural edges, none of whose faces weighs more than $3W/4$. We assume that both K_- and K_+ exist. The cases where one of them or both do not exist are similar. Consider the cycles $X(K_-), X(K_+)$, and $\tilde{C}$. Since $\tilde{C}$ intersects both K_- and K_+, G' is biconnected. See Figure 2 for an illustration.

To establish the bound on the number of edges in G', note that, by choice of $X(K_-)$ and $X(K_+)$, they consist of fewer than $\sqrt{m}$ natural edges each. It remains to bound $|\tilde{C} \cap (K_- - K_+)|$, which consists of two paths in T between $X(K_+)$ and $X(K_-)$. We claim that $i_+ - i_- \le \sqrt{m}$. To see this, observe that by definition of i_- and i_+, for every level $i_- < i < i_+$, the bounding cycle of the unique level-i component intersected by $\tilde{C}$ consists of more than $\sqrt{m}$ natural edges. Since bounding cycles do not share natural edges (Invariant 1) and since there are m natural edges in G, it must be that $i_+ - i_- - 1 < \sqrt{m}$. Combining this with Invariant 2, we conclude that $|\tilde{C} \cap (K_- - K_+)|$ consists of at most $2\sqrt{m}$ natural vertices.

As for the face weights, the weight of a face f' of G' is the total weight of the faces of G that are enclosed in G by the cycle formed by the edges of f'. The faces of G' that correspond to the exterior of $X(K_-)$ and to the interior of $X(K_+)$ have weight at most $3W/4$ by the conditions in Lines 13 and 17. All of the other faces of G' are either enclosed (in G) by $\tilde{C}$ or not enclosed by $\tilde{C}$. Therefore, the weight of each of these faces is at most $3W/4$ since $\tilde{C}$ is a balanced separator. □

Since any simple plane graph has at most three times as many edges as vertices, and since triangulating and making biconnected any plane graph G requires $O(|V(G)|)$ edges, we can restate Lemma 13 in a more general form that does not distinguish natural and artificial vertices.

COROLLARY 2. *Let G be a simple biconnected plane graph with face weights such that no face weighs more than 3/4 the total weight, and such that no face consists of more than 3 edges. There exists a constant c such that the procedure* SIMPLECYCLESEPARATOR *finds a 3/4–balanced simple cycle separator in G whose length is at most $c\sqrt{|V(G)|}$.*

The properties of the procedure GREEDYCYCLESEPARATOR are summarized in the following lemma. See the full version for a description of this greedy algorithm.

LEMMA 14. *Let G be a biconnected planar graph with m edges and face weights summing to W such that no face weighs more than $3W/4$. There exists an $O(m)$ algorithm that finds a balanced simple-cycle separator in G.*

4.5 Efficient Implementation

It is fairly straightforward to implement SIMPLECYCLESEPARATOR in linear time. In this subsection we describe the data structures that support implementing SIMPLECYCLESEPARATOR in sublinear time. Our preprocessing procedure initializes the tree T to be the tree $\bar{T}$ (Lemma 10), the cotree T^* to be the spanning tree of $\bar{G}^*$, rooted at f_∞, whose edges are those not in T. T is represented using an Euler tour tree [24]. T^* is represented using a dynamic tree [46, 15, 50]. The procedure also computes the component tree $\mathcal{K}$ and all level cycles in G. The component tree can be represented by parent pointers. Level cycles are represented by splay trees [47]. For every edge e in $\bar{G}$, let $P^{\mathcal{K}}(e)$ be the path in $\mathcal{K}$ that consists of all components K such that e has one endpoint in $X(K)$ and another endpoint strictly enclosed by $X(K)$. We maintain these paths by storing the first and last components of $P^{\mathcal{K}}(e)$ in an array entry $\mathtt{R(e)}$. This array can be generated in linear time in the preprocessing step.

The full version of the paper describes how this representation allows each required operation to be supported efficiently. Roughly speaking, each basic operation takes $O(\log |E(G)|)$ amortized time. For example, finding a balanced edge separator in T^* (Line 3) is done using a dynamic-tree operation that returns a leafmost edge whose subtree has total weight at least $W/4$. It requires $O(\log |E(G)|)$ amortized time. Another example is finding the levels l and h in Line 7. This is done using the array $\mathtt{R}$ (which is also used for updatind the representation of level cycles in Section 5).

There are $O(\log |E(G)|)$ iterations of the binary search in SIMPLECYCLESEPARATOR. By the arguments in Lemma 13, each iteration scans through $O(\sqrt{|E(G)|})$ levels, performing a constant number of basic operations per level. Therefore, assuming that the auxiliary data structures are given in the representation described above, SIMPLECYCLESEPARATOR can be implemented in $O(\sqrt{|E(G)|}\log^2 |E(G)|)$ time.

5. MAINTAINING THE REPRESENTATION OF REGIONS EFFICIENTLY

To be able to establish the linear running time of the algorithm claimed in Theorem 3, it remains to describe how to maintain the auxiliary data structures that represent the regions and are required by SIMPLECYCLESEPARATOR throughout the recursive calls to RECURSIVEDIVIDE. These include maintaining the embedding of regions, the primal spanning tree T with its monotonicity invariant (Invariant 2), the cotree T^*, the component tree, and the level cycles with their disjointness invariant (Invariant 1).

Consider an iteration of RECURSIVEDIVIDE in which a region R is partitioned into two regions R_0 and R_1 along a cycle separator C. Note that, due to the recursive nature of the algorithm, we may assume that first the representations for R_0 are obtained and the algorithm is invoked recursively on R_0. After the recursive call for R_0 is completed, all changes are undone, and the representations for R_1 are obtained. To achieve linear time for computing an r–division, it is required that the update be done in sublinear time in the size of R. Essentially, the representation can be updated by performing only local operations on the vertices and edges of the separator C. This results in sublinear running time since C consists of $O(\sqrt{|R|})$ edges, and since each operation in the data structures we use takes $O(\log |R|)$ amortized time. The details appear in the full version. Since the cycle can be computed in $O(\sqrt{|R|}\log^2 |R|)$, as established in Section 4.5, the total running time of the algorithm is linear.

6. ACKNOWLEGMENTS

Work by PNK was supported in part by NSF Grant CCF-0964037. Part of the work conducted by SM was done while at Brown University. SM was supported in part by NSF Grants CCF-0964037 and CCF-1111109. Work by CS was conducted while at MIT and supported in part by the Swiss National Science Foundation.

7. REFERENCES

[1] L. Aleksandrov, H. N. Djidjev, H. Guo, and A. Maheshwari. Partitioning planar graphs with costs and weights. *ACM J. Exp. Alg.*, 11, 2006.

[2] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *JAMS*, 3(4):801–808, 1990.

[3] T. Andreae. On a pursuit game played on graphs for which a minor is excluded. *J. Comb. Th. B*, 41(1):37–47, 1986.

[4] L. Arge, G. S. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. *J. Alg.*, 53(2):186–206, 2004.

[5] L. Arge, F. van Walderveen, and N. Zeh. Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In *24th SODA*, pages 901–918, 2013.

[6] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.

[7] P. Biswal, J. R. Lee, and S. Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *JACM*, 57(3), 2010.

[8] S. Cabello. Many distances in planar graphs. *Algorithmica*, 62(1–2):361–381, 2012.

[9] S. Cabello and G. Rote. Obnoxious centers in graphs. *SIAM J. Discret. Math.*, 24(4):1713–1730, 2010.

[10] F. R. K. Chung. Separator theorems and their applications. In *Paths, Flows, and VLSI-Layout, Algorithms and Combinatorics*, pages 17–34. Springer-Verlag, 1990.

[11] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM J. Algeb. Disc. Meth.*, 3:229–240, 1982.

[12] H. N. Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica. Bulgariacae mathematicae publicationes*, 11(4):369–387, 1985.

[13] H. N. Djidjev and S. M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.

[14] D. Eppstein. Dynamic generators of topologically embedded graphs. In *14th SODA*, pages 599–608, 2003.

[15] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Alg.*, 13(1):33–54, 1992.

[16] J. Erickson, K. Fox, and A. Nayyeri. Global minimum cuts in surface embedded graphs. In *23rd SODA*, pages 1309–1318, 2012.

[17] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

[18] E. Fox-Epstein, S. Mozes, P. M. Phothilimthana, and C. Sommer. Short and simple cycle separators in planar graphs. In *14th ALENEX*, pages 26–40, 2013.

[19] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[20] H. Gazit and G. L. Miller. Planar separators and the Euclidean norm. In *SIGAL*, pages 338–347, 1990.

[21] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *J. Alg.*, 5(3):391–407, 1984.

[22] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995.

[23] R. Halin. S-functions for graphs. *J. Geom.*, 8(1-2):171–186, 1976.

[24] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *JACM*, 46(4):502–516, 1999.

[25] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[26] M. Holzer, F. Schulz, D. Wagner, G. Prasinos, and C. D. Zaroliagis. Engineering planar separator algorithms. *ACM J. Exp. Alg.*, 14, 2009.

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

[28] D. B. Johnson and S. M. Venkatesan. Using divide and conquer to find flows in directed planar networks in $O(n^{3/2} \log n)$ time. In *20th Allerton Conference on Communication, Control, and Computing*, pages 898–905, 1982.

[29] K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *38th ICALP*, pages 135–146, 2011.

[30] K. Kawarabayashi and B. A. Reed. A separator theorem in minor-closed classes. In *51st FOCS*, pages 153–162, 2010.

[31] J. A. Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM J. Comput.*, 35(4):882–902, 2006.

[32] P. N. Klein and S. Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

[33] J. Łącki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd FOCS*, pages 599–608, 2012.

[34] J. Łącki and P. Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *19th ESA*, pages 155–166, 2011.

[35] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[36] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[37] A. Maheshwari and N. Zeh. I/O-efficient planar separators. *SIAM J. Comput.*, 38(3):767–801, 2008.

[38] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[39] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *23rd SODA*, pages 209–222, 2012.

[40] S. Mozes and C. Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *18th ESA*, pages 206–217, 2010.

[41] S. A. Plotkin, S. Rao, and W. D. Smith. Shallow excluded minors and improved graph decompositions. In *5th SODA*, pages 462–470, 1994.

[42] B. A. Reed and D. R. Wood. A linear-time algorithm to find a separator in a graph excluding a minor. *TALG*, 5(4), 2009.

[43] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Alg.*, 7:309–322, 1986.

[44] A. L. Rosenberg and L. S. Heath. *Graph Separators, with Applications*. Frontiers in Computer Science. Springer, 2000.

[45] D. B. Shmoys. Cut problems and their application to divide-and-conquer. In *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997.

[46] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[47] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *JACM*, 32(3):652–686, 1985.

[48] D. A. Spielman and S.-H. Teng. Disk packings and planar separators. In *12th SoCG*, pages 349–358, 1996.

[49] S. Subramanian. *Parallel and Dynamic Shortest-Path Algorithms for Sparse Graphs*. PhD thesis, Brown University, 1995. Available as Brown University Computer Science Technical Report CS-95-04.

[50] R. E. Tarjan and R. F. Werneck. Self-adjusting top trees. In *16th SODA*, pages 813–822, 2005.

[51] P. Ungar. A theorem on planar graphs. *J. Lond. Math. Soc.*, 1-26(4):256–262, 1951.

[52] K. G. C. von Staudt. *Geometrie der Lage*. Bauer und Raspe, Nürnberg, 1847.

[53] H. Whitney. Non-separable and planar graphs. *AMS Trans.*, 34(2):339–362, 1932.

[54] C. Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In *52nd FOCS*, pages 37–46, 2011.