TECHNICAL REPORT NO. 309

Approximating Small Separators in Planar Graphs: Trade-offs in Time, Processors, and Separator Size

by

Gregory E. Shannon and Fang Wan

June 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

Approximating Small Separators in Planar Graphs: Trade-offs in Time, Processors, and Separator Size[*]

Technical Report #309

Gregory E. Shannon
Fang Wan

Department of Computer Science
Indiana University
Bloomington, Indiana 47405

June 1990

Abstract

This report describes progress in efforts to design an NC algorithm for finding $O(\sqrt{n})$-size separators in n-vertex planar graphs. We present deterministic algorithms which all use n processors and can find successively smaller and smaller sub-linear-size separators. For each decrease in the separator size (starting with n vertices), the time is increased by a $\log n$ factor. In particular, our algorithm can find an $O\left(n^{\frac{1}{2}+\frac{1}{2}\alpha^i}\right)$-size cyclic separator in a triangulated planar graph using n processors and $O(\log^{i+2} n)$ time, for a constant $\alpha < 1$.

[*]This research has been supported in part by Hewlett-Packard's Faculty Development Program and NSF Grant CCR-89-09535.

1 Introduction

The problems of breadth-first-search (BFS) and single-source shortest paths are of fundamental importance in computational graph theory [1, 17, 22, 26]. For the important case of planar graphs, algorithms for finding $O(\sqrt{n})$-size separators have proven to be important tools in solving these two problems [5, 17, 21], and are also of independent importance [6, 7, 17, 18, 21]. (A vertex set S of an n-vertex graph G *separates* G if no connected component in $G - S$ has more than $2n/3$ vertices.)

Though these three problems have (near-)linear-time sequential algorithms [1, 16, 18, 22], none has an NC (polylog time) parallel algorithm which uses a linear number of processors. The NC algorithm for shortest paths in general digraphs still requires n^3 processors [13]. Therefore, there is a great deal of progress yet to be made in finding fast parallel algorithms with near-linear processor-time products for these well-studied problems.

We present here deterministic linear-processor NC algorithms for finding $O(n^\epsilon)$-size cyclic separators in n-vertex triangulated planar graphs where $\frac{1}{2} \leq \epsilon < 1$. To find an $O(n^{\epsilon_i})$-size separator, $\epsilon_i = \frac{1}{2} + \frac{1}{2}\alpha^i$, our algorithm uses $O(\log^{i+2} n)$ time and only n processors, for some constant $\alpha < 1$. The constant α is equal to $1 - 1/(\gamma M)$ where $\gamma > 1$ and n^M is the number of processors needed to multiply two $n \times n$ matrices over the ring of integers in $O(\log n)$ time. The value of M is bounded below by 2 and is currently bounded above by 2.376 [4]. If more than $p > n$ processors are available, then the constant α is closer to 0 ($\alpha = 1 - p/(2M)$) and smaller separators can be found in less time. The algorithms have the same asymptotic performance if $M = 3$ as with straightforward matrix multiplication. Our algorithms are for the concurrent-read concurrent-write parallel RAM (CRCW PRAM) model of computation. See [13] for more background on the PRAM.

The primary technique for our algorithm is to decompose the graph into r- divisions using the successively smaller separators. An r- *division*, as introduced by Frederickson [5], is a division of a graph's n vertices into $\Theta(n/r)$ connected regions such that each is of size $O(r)$ and shares (or has a boundary of) at most $O(\sqrt{r})$ vertices with other regions. (In Section 2 we give a more restricted definition of a division.) Frederickson assumes the recursive subdividing is being done with $O(\sqrt{n})$-size separators. We extend his technique to work with $O(n^{\epsilon})$-separators, $\frac{1}{2} \leq \epsilon < 1$. Of independent interest, we also show how to use any separator to find well-formed partitions of faces so that each region is well connected.

Wan [28] has shown how to find an $O(n^{.79})$-size cyclic separator with n processors. With our generalization of Frederickson's technique, we find a well-formed decomposition of a planar graph with Wan's algorithm. By considering only the boundary vertices in the decomposition, the skeleton, we apply to the skeleton Miller's n^M-processor NC algorithm [18, 10] for finding $O(\sqrt{n})$-size cyclic separators. This produces an even smaller separator than Wan's algorithm. This process can be repeated to produce algorithms which find smaller and smaller separators. In $O(\log\log n)$ iterations (or algorithms), we have an algorithm which finds an $O(\sqrt{n})$ cyclic separator using only n processors. Unfortunately, this final algorithm uses $O(\log^{O(\log\log n)} n)$ time — not an NC algorithm.

Previous algorithms use more processors, are randomized, have worse processor-time products, or produce larger separators.

The original deterministic NC algorithm for finding small cyclic separators in planar graphs is by Miller [18]. It uses $O(\log^2 n)$ time and n processors plus as many processors as are needed to find a BFS tree in a general graph with as many vertices as the input planar graph has faces. Gazit and Miller [10] have shown how to find a BFS tree in a general graph using n^M processors. (M is defined above.) For an triangulated n-vertex planar graph, Miller's algorithm produces a cyclic separator with at most $2\sqrt{2n}$ edges. If the graph is not triangulated but is biconnected, has f faces, and no face has more than b edges in its boundary, the algorithm produces a cyclic separator with $2b\sqrt{f}$ edges. Our algorithms use radically fewer processors but produce larger separators.

Gazit and Miller [9, 8] have a randomized algorithm that uses $O(\log^2 n)$ expected time and $n + f^{1+\rho}$ processors, for constant $\rho > 0$, on an n-vertex biconnected planar graph with f faces. The bounds on the separator size are the same as for Miller's deterministic algorithm. In comparison, our algorithm is deterministic and has a smaller processor time product. Our algorithms can find an $O(n^{\frac{1}{2}+\epsilon})$-size separator in polylogarithmic time for any constant $\epsilon > 0$ with n processors. Therefore, our algorithms can use fewer processors in exchange for larger separators. For an $O(\sqrt{n})$-size separator, our algorithm has a better processor-time product. For a triangulated graph, f is $O(n)$, and $n^{1+\rho}$ asymptotically dominates $O(log^{O(\log\log n)}n)$.

An algorithm by Wan [28] finds an $O(n^{.79})$-size cyclic separator in a triangulated planar graph in $O(\log^2 n)$ time using n processors. The bound on the separator increases as discussed above when the number of faces and the maximum boundary size implies a larger separator, as in Miller's original work. Our results were originally motivated by Wan's algorithm — primarily by the fact that a sub-linear size separator could be found with n processors in polylog time. Our results enable us to find much smaller separators with relatively small increases in the time complexity.

A major reason that separators have already proven to be a significant paradigm for designing parallel algorithms on planar graphs is that Pan and Reif have shown how to solve a large number problems on planar graphs using polylog time and $n^{1.5}$ processors [21]. Actually, the number of processors needed is $n^{3\epsilon}$ when n^{ϵ} is the size of separator available. Therefore, our improved algorithms offer the first deterministic algorithms for significantly reducing the processor complexity of problems by capitalizing on Pan and Reif's work. In particular, deterministic NC algorithms immediately follow for BFS on planar graphs which use fewer than n^2 processors. In fact, with our separator algorithms, we can derive algorithms for finding single source shortest paths in planar graphs using polylog time and $n^{\epsilon M}$ processors

[24]. Miller and Naor [19] have shown how to use separators to develop efficient NC parallel algorithms for various flow problems on planar graphs. Again, the bottleneck in reducing the algorithms' processor complexities is the difficulty in finding small separators.

In the next section we provide some standard definitions and basic properties of planar graphs. In Section 3, we present the notion of a well-formed separator for regions of faces and describe how to find one, given a separator algorithm for triangulated graphs. In Section 4, we show how to efficiently subdivide a planar graph using an algorithm for finding cn^{ϵ}-size separators. In Section 5, we show how to use the subdivision from Section 4 to find a smaller separator. In Section 6, we discuss how the approach in Sections 4 and 5 can be iterated to produce algorithms with successively smaller and smaller separators. In Section 7, we close by discussing extensions and implications of our work and the difficulties which are keeping us from designing a linear-processor NC algorithm for finding $O(\sqrt{n})$-size separators.

2 Preliminaries

For graph terms not defined here, see the textbooks on graph theory by Harary [Har69] or Bondy and Murty [BM76]. We assume that any planar graph used in this paper is represented as a combinatorial embedding with ordered adjacency lists as in [16].

The *boundary* of a face is the set of edges which separates that face from all other faces. A boundary is simple if it is a simple cycle. If the graph has no self-loops and is biconnected, then each face's boundary is simple [27]. A face's *size* is the number of edges on its boundary. Two faces are *edge connected* iff they share an edge in their boundaries. A set of faces is edge connected if each pair of faces is directly or indirectly edge connected.

A *region* in a planar graph is an edge-connected set of simple faces. The boundary of a region is those edges which are adjacent to only one face in the region. See Figure 1 for an example of a region. The weight of a region is the sum of the weights of the faces in the region. A set of edges S separates a set of faces into two sets A and B iff edges in S are the only boundary edges shared between faces in A and faces in B. S must also be a simple cycle or a collection of simple chains. An edge set separates a region if each of the resulting regions has a weight of no more than two thirds of the original region's weight. Figure 2 shows an example of weights and a separator for a region of faces. We assume that only faces have weights, though the algorithms here work as just as well when edges and vertices also have weights. Without loss of generality, we assume the sum of the face weights is usually normalized to 1.

The dual of a region R is very similar to the dual of the graph implied by the edges in R [11]. For a region, however, there is one vertex for each face in R, and if two regions share a boundary edge, then there is an edge between the corresponding vertices. See Figure 3 for an example.

In the following sections, we assume that $Se(c, \epsilon)$ is an algorithm for finding a cyclic separator S in an n-vertex simple maximal planar graph such that S contains at most cn^{ϵ} edges.

3 Well-formed Separators

In this section we present a technical algorithm which uses a cyclic separator for a maximal planar graph to find an edge separator for a region of triangular faces. The basic idea is that collapsing each connected component of the region's boundary into a single vertex produces a triangulated graph after simple one- and two-edge separated components are removed using a spanning tree on the dual. This will enable us to construct a division of a planar graph where the number of divisions matches the number of faces. Frederickson's techniques do not guarantee this important property.

Given a region R of weighted faces, the approach is to collapse each connected component of R's boundary vertices into a single vertex. As vertices are merged, duplicate edges and self-loops are *not* removed and the relative embedding of the edges is maintained around each new supervertex. A spanning tree on the dual of this collapsed graph R' is then used either to find a simple 1- or 2-edge separator S for R' or to remove 1- and 2-edge separated components while properly removing duplicate edges and self-loops and updating the weights of adjacent faces. The cyclic separator algorithm for maximal graphs is then applied to R' to find a separator S. Because of the way R' is constructed, S is also an edge-separator for R. Figure 4 shows how this algorithm would proceed on an example graph. Figure 5 presents the algorithm more formally.

Before we prove the correctness and complexity of the algorithm *Well- formed_Separator*, we explain how Steps 4 and 5 are implemented.

Self-loops can be detected in constant time and quickly eliminated. Let e be a self-loop in R', and let f and g be the faces adjacent to e. Assume that g is the parent of f in T. Let w_f be the weight of the subtree in T rooted at f. If $\frac{1}{3} \leq w_f \leq \frac{2}{3}$, then e is a separator. If $w_f < \frac{1}{3}$, then delete e and all the faces covered by the subtree rooted at f and set g's weight to it's initial weight (not w_g) plus w_f. If $w_f > \frac{2}{3}$, then delete e and all the faces *not* covered by the subtree rooted at f and set f's weight to it's initial weight (not w_f) plus w_g. Though any vertex might have many self-loops and self-loops can be nested, simple treefix types of operations [13, 20] can quickly determine which faces to merge and delete and the final face weights in $O(\log n)$ time and n processors.

For duplicate edges, the approach is similar to that used for self-loops. To determine which pairs of vertices have duplicate edges, sort the remaining edges in R' based on their endpoints. Let x and y be two vertices in R' with duplicate edges $e_1, e_2, \ldots, e_k$ as ordered around x (or y). With a parallel prefix approach and the subtree weights in T, it straightforward to determine if any pair of duplicate edges between x and y form a two-edge separator. If no pair does, then there exists a pair of edges e_i and e_{i+1} (really $e_{(i+1 \bmod k)+1}$) such that the weight between e_i and e_{i+1} is more than $\frac{2}{3}$. Therefore, collapse all the faces between e_{i+1} and e_i into one face as demonstrated in Figure 6. Now there are only two duplicate edges between x and y and these two edges bound at least one face. As with the self-loops, treefix types of operations can quickly determine which faces are to be merged and which duplicates have been eliminated.

Each face of size 2 now chooses an adjacent face with weight less than $\frac{1}{3}$ and merges with it. However, up to three size-2 faces might merge with one size-3 face and produce a new face with a weight of more than $\frac{2}{3}$. If such is the case, then a 3-edge cyclic separator can be found in constant time. Therefore, all of the multiple edges have been removed and each face has a weight less than $\frac{1}{3}$.

The lemma below shows that an edge separator for R need contain only edges adjacent to two faces in R.

Lemma 1 *Let R be a region of triangular faces. If S is an edge separator of R, then the edges in S adjacent to exactly two faces in R also form an edge separator of R.*

Proof. Let e be an edge in S which is adjacent to no faces in R. Such an edge can not exist since all edges in R must be on the boundary of one of R's member faces by definition. Let e be an edge in S which is adjacent to exactly one face in R. Its removal from S does not directly or indirectly affect the edge-connectivity of any of R's faces. Therefore, edges not adjacent to two faces in R don't help separate any faces in R, and the lemma holds. ■

We are now ready to prove the correctness and complexity of the algorithm *Well-form_Separator.*

Theorem 1 *Given a region R of f triangulated faces, the algorithm* Well- formed_Separator(c, ϵ) *finds a simple edge-separator S for R such that S has at most $c\left(\frac{f+4}{2}\right)^{\epsilon}$ edges. The algorithm uses $O(T_{Sep(c,\epsilon)}(f) + \log f)$ time and $P_{Sep(c,\epsilon)}(f) + f$ processors.*

Proof. We first verify that S is a separator of R. Each edge e' in R' corresponds to some edge e in R. (Vertices in R are mapped onto vertices in R'.) Since no face in R or R' ever has a weight of more than $\frac{2}{3}$, if S is a cyclic separator in R', then $S \cup \mathit{Boundary}(R)$ is an edge-separator for R. However, Lemma 1 shows that all the edges in $\mathit{Boundary}(R)$ are unnecessary if S only needs to be an edge separator for R. S must be a collection of chains and cycles in R since S is a cyclic separator in R' and since if S has a vertex of degree greater than two in R, then it also has one in R'. Therefore, S is a simple edge separator for R.

Since R' is a maximal planar graph with at most f faces, it has $(f+4)/2$ vertices by Euler's theorem [11]. Therefore, S has at most $c\left(\frac{f+4}{2}\right)^{\epsilon}$ edges by the definition of $\mathit{Sep}(c,\epsilon)$.

Both R and R' have $O(f)$ vertices. Since R has at most three vertices per face, $n \leq 3f$. Therefore, all of the subalgorithms except $\mathit{Sep}(c,\epsilon)$ use at most f processors and $O(\log f)$ time. This follows from well-known algorithms for finding a dual, treefix computations, arbitrary spanning trees, and parallel prefix [25, 3, 2, 15, 14, 20, 12]. $\mathit{Sep}(c,\epsilon)$ uses $O(T_{\mathit{Sep}(c,\epsilon)}(f))$ time and $P_{\mathit{Sep}(c,\epsilon)}(f)$ processors since $f = 2n-4$. ■

4 (r,ϵ)-Divisions

In this section we show how to extend Frederickson's algorithm for finding r-divisions in planar graphs using $O(\sqrt{n})$ separators [5]. In particular, we use $O(n^{\epsilon})$-size separators, for constant $\epsilon > 0$, focus on faces and edges instead of vertices, and produce only well-formed regions. An (r,ϵ)-*division* of a region of f triangulated faces is partition of the faces into $\Theta(f/r)$ subregions each with at most r faces and $O(r^{\epsilon})$ boundary edges. By boundary edges here, we do not mean the boundary of the region, but the boundaries between subregions. Each boundary edge is adjacent to two different subregions. Figure 7 gives an example of a division.

The algorithm $\mathit{Division}(c,\epsilon)$ below in Figure 5 partitions a region of faces into subregions of triangular faces by recursively selecting edge separators to form boundaries. The subregions form an (r, ϵ)-division of the region. Unlike Frederickson's approach, we ensure that each subregion of faces is edge connected. This simplifies the algorithm and adequately controls the number of subregions. In the first phase (Steps 1-5), the small subregions are formed. Each face is assumed to have equal weight. In the second phase, the size of each region's boundary is brought under control. This is done by setting a face's weight equal to the number of its boundary edges that are also boundary edges for the subregion. The algorithm *Sep*(c, ϵ) is then applied to the region with the new face weights. Note that $Weight(R)$ is the sum of the weights of the faces in R, *Inside*(S, R) (*Outside*(S, R)) is the set of R's faces to the right (left) of S or inside (outside) the cycle separator found in the algorithm *Well-formed_Separator*, *Bound_Size*(R) is the number of R's boundary edges. Figure 8 gives an example of how the first 5 steps of *Division* might proceed.

We spend most of the rest of this section characterizing the number of boundary edges and regions formed by the algorithm *Division.*

Lemma 2 *Before Step 6 in the algorithm* Division(c, ϵ) *there are at most* $\frac{c'cf}{r^{1-\epsilon}}$ *edges in Boundary where* $c' \leq \frac{3^{2\epsilon}\sqrt{3}}{2^{\epsilon}(1+2^{\epsilon}-3^{\epsilon})}$ *for* $\frac{1}{2} \leq \epsilon < 1$.

Proof. Let $B(f)$ be the number of edges in *Boundary* before Step 6. From the algorithm, it is easy to see that $B(f)$ must satisfy the recurrence below.

$$\begin{aligned} B(f) &= 0, && f \leq r \\ B(f) &= c\left(\tfrac{f+4}{2}\right)^{\epsilon} + B(\alpha f) + B((1-\alpha)f), && f > r,\ \frac{1}{3} \leq \alpha \leq \frac{1}{2} \end{aligned}$$

We conjecture that for constants c' and $d > 0$,

$$B(f) \leq \frac{c' f}{r^{1-\epsilon}} - d f^{\epsilon}. \tag{1}$$

We prove Equation 4.1 by induction on f. The basis case ($f \leq r$ and $f \geq r/3$) holds when

$c' \geq \sqrt{3}d$. For the induction case, we first expand $B(f)$ based on the induction hypothesis.

$$B(f) = c\left(\frac{f+4}{2}\right)^{\epsilon} + B(\alpha f) + B((1-\alpha)f) \tag{2}$$

$$\begin{aligned} B(f) &\leq c\left(\frac{3f}{2}\right)^{\epsilon} + B(\alpha f) + B((1-\alpha)f) &(3)\\ &\leq c\left(\frac{3f}{2}\right)^{\epsilon} + \left[\frac{c'\alpha f}{r^{1-\epsilon}} - d(\alpha f)^{\epsilon}\right] + \left[\frac{c'(1-\alpha)f}{r^{1-\epsilon}} - d((1-\alpha)f)^{\epsilon}\right] &(4)\\ &\leq \frac{c'f}{r^{1-\epsilon}} + c\left(\frac{3f}{2}\right)^{\epsilon} - df^{\epsilon}(\alpha^{\epsilon} + (1-\alpha)^{\epsilon}) &(5)\end{aligned}$$

In order for Equation 1 to hold, we must show that the RHS of Equation 5 is bounded above by the RHS of Equation 1. Therefore, we must show that Equation 6 below holds.

$$c\left(\frac{3}{2}\right)^{\epsilon} - d(\alpha^{\epsilon} + (1-\alpha)^{\epsilon}) \leq -d \tag{6}$$

Rearranging terms, we have Equation 7.

$$d \geq \frac{c3^{e}p}{2^{\epsilon}(\alpha^{\epsilon} + (1-\alpha)^{\epsilon} - 1)} \tag{7}$$

To determine the largest lower bound on d, we must determine for what α the expression $\alpha^{\epsilon} + (1-\alpha)^{\epsilon}$ is minimized if $\frac{1}{3} \leq \alpha \leq \frac{1}{2}$. Looking at the derivatives, it is easy to show that it is minimized when $\alpha = \frac{1}{3}$ for any ϵ when $\frac{1}{2} \leq \epsilon < 1$. Subsituting $\frac{1}{3}$ for α and rearranging, we have Equation 8.

$$d \geq \frac{c3^{2\epsilon}}{2^{\epsilon}(1 + 2^{\epsilon} - 3^{\epsilon})} \tag{8}$$

As stated in the basis step, $c' \geq \sqrt{3}d$, and the lemma holds. ∎

Lemma 3 *At Step 13 in the algorithm* Division(c, ϵ)*, there are at most* $\frac{f}{r}\left(\frac{4}{3} + 3c'\right)$ *regions and each region has a boundary with of most* $9cr^{\epsilon}$ *edges.*

Proof. Let t_i be the number of regions with i boundary edges just before Step 6. For a region with $i > 9cr^{\epsilon}$ boundary vertices, the second while loop splits it into at most $i/(cr^{\epsilon})$

regions, each with at most $9cr^\epsilon$ boundary vertices. To see this, consider the recurrence below which counts how many regions are generated. It is bounded above by $(i - 2cr^\epsilon)/(cr^\epsilon) - 1$.

$$\begin{array}{rcll} R(i) & = & 0, & i \leq 9cr^\epsilon \\ R(i) & = & 1 + R(\alpha f + cr^\epsilon) + R((1-\alpha)f + cr^\epsilon), & \text{otherwise, } \dfrac{1}{3} \leq a \leq \dfrac{1}{2} \end{array}$$

There will be at most cr^ϵ new boundary edges per new region. Note that Lemma 2 shows us that there are at most $\Sigma_i(it_i) \leq 3\,c'c\,f/r^{1-\epsilon}$ boundary edges since each face has a boundary of size 3. Therefore, the number of new regions is at most $\Sigma_i\left(\dfrac{i}{cr^\epsilon}\,t_i\right) \leq 3\,c'f/r$. The total number of regions is then $\frac{f}{r}\left(\frac{4}{3} + 3c'\right)$. ∎

Theorem 2 *Given a region R of f triangulated faces, the algorithm* Division(c,ϵ) *finds an (r,ϵ)-division of R. There are at most $\frac{n}{r}\left(\frac{4}{3} + 3c'\right)$ regions and each region has a boundary with of most $9cr^\epsilon$ edges. The algorithm uses $O(\log n \cdot T_{Sep(c,\epsilon)}(f) + \log f)$ time and $P_{Sep(c,\epsilon)}(f) + f$ processors.*

Proof. Given a region R of f triangulated faces, the partition of faces returned by the algorithm *Division*(c,ϵ) is an (r,ϵ)-division of R. By Lemma 3, the bounds hold on the number of regions and boundary vertices per region in the returned partition.

In the first phase (through Step 5) the while-loop is executed only $O(\log n)$ times since the number of faces in each new subregion ($Inside(R)$ and $Outside(R)$ in Step 11) has at least one-third fewer faces than the region R from which it was formed. The while-loop in the second phase (starting at Line 6) also only executes $O(\log n)$ times since the boundary of each region is being reduced by a constant factor in each iteration.

In each of the while-loops, the calls to *Well-formed_Separator* use $O(T_{Sep(c,\epsilon)}(f) + \log f)$ time and $P_{Sep(c,\epsilon)}(f) + f$ processors by Theorem 1. Since R has at most three vertices per face, $n \leq 3f$. Therefore, all of the other steps use at most f processors and $O(\log f)$ time. This follows from the algorithms for treefix computations, arbitrary spanning trees, and parallel prefix as in Theorem 1. Note that each partition P_i requires only $O(f)$ space/processors since each edge in R is in at most two subregions in P_i. Therefore, the algorithm uses $O(\log f T_{Sep(c,\epsilon)}(f) + \log^2 f)$ time and $P_{Sep(c,\epsilon)}(f) + f$ processors. ■

5 Improved Separators

In this section we show how to use an (r, ϵ)-division to find a separator with significantly fewer edges than cn^ϵ. Our approach follows that of Wan [28] in that after forming a sparse partition of the faces into well-connected regions, Miller's brute force algorithm is applied to the skeleton. Our advantage over Wan's approach is that with an (r, ϵ)-division the boundary of each superface in the skeleton is sublinear in the number of original faces used to form it.

The algorithm *Improved_Separator*$(c, \epsilon)[G]$ assumes there is an algorithm *Sep*(c, ϵ) available for finding cyclic separators with at most cn^ϵ edges in n-vertex triangulated planar graphs. First, an $(n^{1-1/M}, \epsilon)$-division P is found using the algorithms *Sep*(c, ϵ) and *Division*. G is passed to *Division* as a region containing all of G's faces with equal weights. The *skeleton* G' of G with respect to P is just G's subgraph induced by the boundary edges in P. Miller's separator algorithm is then applied to G' if it is biconnected. Figure 9 below summarizes the algorithm.

If G' is only connected or is disconnected we can either find a small separator or apply Miller's algorithm to an appropriate biconnected component. For these degenerate cases, consider a spanning tree T of the dual of G'. We find a *balancing point* v in T such that the weight w_v of the subtree rooted at v in T is at least $\frac{1}{3}$ and none of it's children have a greater than $\frac{1}{3}$. If v is a cut-point in the dual (either its boundary is not connected or non-cyclic) and $w_v \leq \frac{2}{3}$, then there is a subgraph of its boundary which is a cyclic separator for G'. See Figure 10 for an example of a non-connected skeleton. If there is no balance point with both properties, then a biconnected subgraph of G' must be found. Figure 11 gives an example of how this can proceed. From v in T, each path to a leaf in T is followed until a cut-point in the dual is found or the leaf is reached. If a cut-point x is found, then no exploring beyond x is done. The subgraph of faces covered by the subtree rooted at x in T are removed. x's weight is updated to w_x if x is a descendent of v. If x is an ancestor of v, then x becomes the exterior face and its weight is set to $1-w_x$. Once this biconnected subgraph has been formed, apply Miller's algorithm to it. The approach for non-biconnected components is similar to the one discussed by Miller [18].

The theorem below characterizes the correctness and efficiency of the algorithm *Improved_Separator*.

Theorem 3 *Given a triangulated planar graph G and an algorithm* Sep(c, ϵ), *the algorithm* Improved_Separator(c, ϵ) *finds a cyclic separator for G with at most $C\,c\,n^{\epsilon'}$ edges, for a constant $C > 0$ and where $\epsilon' = \frac{1}{2M} + \epsilon\left(1 + \frac{1}{M}\right)$. The algorithm uses $O(\log n \cdot T_{Sep(c,\epsilon)}(n) + \log^2 n)$ time and $P_{Sep(c,\epsilon)}(n) + n$ processors.*

Proof. First, we prove the algorithm's correctness. If G' is biconnected, then Miller's algorithm returns a cycle that also separates G. If G' is only connected or is disconnected then a balance point always exists. If the balance point is an cut-point in the dual with $\frac{1}{3} \leq w_v \leq \frac{2}{3}$, then it is straightforward to show indirectly that some subgraph of the face's boundary is a separator. Which part depends on where the root is relative to the cycles in the face's boundaries. See figure 10 for an example of this. If the balance point is not a cut point or doesn't have the appropriate separating property, then it is straightforward to show that collapsing the faces hanging off of cut points will produce the correct biconnected component in which to find a separator. The example in Figure 11 carries out this intuition and is also discussed by Miller [18].

We now bound the size of S when G' is biconnected. G' has as many faces as P has regions, and the bound on the face boundary sizes in G' is the same as that on the region boundary sizes for P. In Miller's algorithm [18, 9, 8], a graph with F faces and a maximum face boundary size of B has a cyclic separator of size $2\,B\sqrt{F}$. By Lemma 2 and since G is triangulated, we have the following for a constant $C > 0$.

$$\begin{aligned}
B &= 9\,c\left(f^{1-\frac{1}{M}}\right)^{\epsilon} \\
&\leq 9\,c\left((2\,n)^{1-\frac{1}{M}}\right)^{\epsilon} \\
F &= f^{\frac{1}{M}}\left(\frac{4}{3}+3\,c'\right) \\
&\leq (2\,n)^{\frac{1}{M}}\left(\frac{4}{3}+3\,c'\right) \\
2\,B\sqrt{F} &\leq 2\cdot 9\,c\left((2\,n)^{1-\frac{1}{M}}\right)^{\epsilon}\sqrt{(2\,n)^{\frac{1}{M}}\left(\frac{4}{3}+3\,c'\right)} \\
&\leq C\,c\,n^{\epsilon'}
\end{aligned}$$

Given the constraints on boundary and region sizes, any trivial separator for the non-biconnected cases will have only $O(r^{\epsilon})$ edges since they involve the boundary of only one face which is much less than $O(\sqrt{n})$.

The main constraint in the processor complexity is applying Miller's algorithm. It uses $n + f^{M}$ processors where f is the number of faces. However, since G' has only $O(n^{1/M}$ faces, only $O(n)$ processors are needed. All of the other algorithms use only n processors by Theorem 2 and from the algorithms discussed below concerning the time complexity.

The first part of the time complexity comes from Theorem 2. The second part is dominated by Miller's algorithm. It is the only subalgorithm we use that require $O(\log^2 n)$ time. As discussed in the proofs of Theorems 1 and 2, only $O(\log n)$ time is necessary for the other operations which reduce finding a dual, finding a spanning tree, parallel prefix, and treefix operations. ∎

Corollary 1 *Given a triangulated planar graph G and an algorithm* Sep(1, 1)*, the algorithm* Improved_Separator *can find a cyclic separator for G with at most $2\sqrt{2}\,n^{\epsilon}$ edges, where $\epsilon = \frac{1}{2M} + \left(1 - \frac{1}{M}\right) = \frac{1}{2} + \frac{1}{2M}$. The algorithm uses $O(log^2 n)$ time and n processors.*

Proof. For the algorithm *Sep*$(1,1)$ use the separator algorithm by Shannon [23]. It finds a cyclic separator in a planar graph with no bound on its size. It runs in $O(\log n)$ time and n processors.

Note that for this case, *Improved_Separator* executes only the first phase of *Division* when computing the partition. Therefore, P has at most $\frac{4}{3}f/r$ regions. Each region has a boundary size of r. By selecting r to be $f^{1-1/M}/x$ for some constant $x > 1$, it easy to see from the the discussion in the proof of Theorem 3 that we can force the separator to have at most $2\sqrt{2}\, n^{\epsilon}$ edges.

Since *Sep*$(1,1)$ uses $O(\log n)$ time and n processors, the complexity follows immediately from the discussion in the proof of Theorem 3. ■

This corollary matches Wan's results [Wan90]. For $M = 3$, the corollary implies a separator algorithm with $\epsilon = \frac{5}{6}$. For $M = 2.376$, $\epsilon < 0.79$. This corollary provides us with the basis for iteratively improving the separator size using the algorithm *Improved_Separator*. We discuss how these iterations converge in the next section.

6 Iteratively Improved Separators

In this section we analyze the repeated applications of *Division* and *Improved_Separator* in finding better and better separators.

Let Sep_i be the separator algorithm formed by i levels of the algorithm *Improved_Separator*. For example, Sep_2 works by assuming an $O(n^{\frac{1}{2}+\frac{1}{2M}})$ separator algorithm Sep_1 exists so that it can call *Division*$(n^{1-1/M}, \frac{1}{2} + \frac{1}{2M})$. However, we know that the algorithm Sep_1 for finding an $O(n^{\frac{1}{2}+\frac{1}{2M}})$-size separator also calls *Division* which in turn uses Shannon's linear-size separator algorithm (Sep_0).

As the basis case, we have a separator with at most n edges. Sep_1 produces a separator with at most $2\sqrt{2}n^{\frac{1}{2}+\frac{1}{2M}}$ edges. Sep_2 produces a separator with at most $C{\cdot}2\sqrt{2}\, n^{\frac{1}{2M}+[\frac{1}{2}+\frac{1}{2M}](1-\frac{1}{M})}$

edges. For Sep_3 and so on, does the size of the separator converge? If so, to what and at what rate? Let each successive application of *Improved_Separator* be an *iteration*. We contend that in $O(\log\log n)$ iterations, the size converges to $O(\sqrt{n})$. By Theorems 2 and 3, each iteration adds a $\log n$ factor to the time and only n processors are ever needed.

Before we can effectively address the convergence of the separator's size, we must resolve the conflict of a growing constant factor and a decreasing exponent. The constant factor increases by C from the second iteration onward and the exponent decreases by some amount. In Theorem 3 it is easy to show that $\epsilon' < \epsilon$ if $\epsilon > \frac{1}{2}$.

Let $s(i)$ be the exponent on the separator size after i iterations. By Theorem 3 and its Corollary,

$$\begin{aligned} s(0) &= 1 \\ s(i+1) &= \frac{1}{2M} + s(i)\left(1 - \frac{1}{M}\right). \end{aligned}$$

Rearranging the recurrence produces $s(i) = \frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{M}\right)^i$.

Let $C(i)$ be the constant on the separator size after applying the algorithm *Improved_Separator* i times. Therefore, by Theorem 3 and its Corollary,

$$\begin{aligned} C(1) &= 1 \\ C(i+1) &\leq C \cdot C(i). \end{aligned}$$

Rearranging the recurrence produces $C(i) \leq C^{i-1}$. Therefore, Sep_i can find a separator with at most $C(i) \cdot 2\sqrt{2}\, n^{s(i)}$ vertices.

However, if we are conservative in how much the exponent improves, then we can cancel out the increase in the constant. The recurrence we intend to satisfy for the size of the exponent and an *unchanging constant* is

$$s'(i) = \frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{\alpha M}\right)^i \quad \text{for constant } \alpha > 1. \tag{9}$$

To understand how the constant of C is removed, consider the potential decrease in the exponent as implied by Theorem 3. If e is the initial exponent and e' is the new one, then the potential decrease is

$$\begin{aligned} e - e' &= e - \frac{1}{2M} - e\left(1 - \frac{1}{M}\right) \\ &= \frac{e}{M} - \frac{1}{2M}. \end{aligned} \tag{10}$$

Fortunately, $s'(i) - s'(i+1)$ is all the decrease that is needed, which is

$$\begin{aligned} s'(i) - s'(i+1) &= \frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{\alpha M}\right)^i - \frac{1}{2} - \frac{1}{2}\left(1 - \frac{1}{\alpha M}\right)^{i+1} \\ &= \frac{1}{2}\left(1 - \frac{1}{\alpha M}\right)^i \left(\frac{1}{\alpha M}\right) \\ &= \frac{s'(i) - \frac{1}{2}}{\alpha M}. \end{aligned} \tag{11}$$

From Equations 10 and 11, the exponent remaining to cancel out the constant C is

$$\begin{aligned} a(i) &= \left[s'(i) - \left(\frac{1}{2\,M} + s'(i)\left(1 - \frac{1}{M}\right)\right)\right] - [s'(i) - s'(i+1)] \\ &= \frac{s'(i)}{M} - \frac{1}{2M} - \frac{s'(i) - \frac{1}{2}}{\alpha M} \\ &= \frac{1}{2\,M}\left(1 - \frac{1}{\alpha}\right)\left(1 - \frac{1}{\alpha M}\right)^i \end{aligned} \tag{12}$$

The question now is, what is the maximum i such that $n^{a(i)} \geq C$? With simple algebra, we see that

$$i \;\geq\; K = \frac{\log\log n - \log\left(2\,M\,\frac{\alpha}{\alpha-1}\,\log C\right)}{\log\left(\frac{\alpha M}{\alpha M - 1}\right)}. \tag{13}$$

The main result of this paper now follows.

Theorem 4 *For a triangulated n-vertex planar graph, i iterations of the algorithm* Improved_Separator *finds a cyclic separator with at most $2\sqrt{2}n^{s'(i)}$ vertices for some α if $i \leq K$ in Equation 13. The ith algorithm uses $O(\log^{i+1} n)$ time and n processors.*

Proof. The correctness of this theorem follows from the above discussion and Theorems 2 and 3 for *Division* and *Improved_Separator*. The time and processor complexities follow from Shannon's separator algorithm [23] and Theorems 2 and 3. ∎

If we iterate *Improved_Separator* K times as implied in Equation 13, then $c\,n^{s'(K)}$ is the best bound possible on the separator size. How close is this to $c\sqrt{n}$? Are we within a constant factor C^*? The ratio below is the factor by which the Kth separator is larger than $c\sqrt{n}$ and be found with simple algebra from Equations 9 and 13.

$$C^* \;=\; \frac{c\,n^{s'(K)}}{c\sqrt{n}} \;=\; n^{\frac{1}{2}\left(1-\frac{1}{\alpha M}\right)^K} \;=\; C^{2\,M\,\frac{\alpha}{\alpha-1}} \tag{14}$$

Since this is a constant, in $O(\log\log n)$ iterations an $O(\sqrt{n})$-size cyclic separator is produced. From Equation 14 and Theorem 4, we have the theorem below.

Theorem 5 *For a triangulated n-vertex planar graph, K iterations of the algorithm* Improved_Separator *finds an $O(\sqrt{n})$-size cyclic separator using n processors and $O(\log^{d\log\log n} n)$ time, for some constant $d > 0$.*

Remark. The constant factors used in Theorems 3 and 5 are not small. C is about 100. Therefore, the $O(\sqrt{n})$-size separator can have a constant of about 100^{10}. However, we have not tried to minimize the constant in making these estimates. We assumed $\alpha = 2$. Also, we have chosen clarity in the presentation over smaller constants. We have no doubt that much smaller constants can be found.

Remark. Our analysis of how the bound on the separator's exponent decreases is rather crude in that we grossly underestimate the gains made in the early iterations. A different analysis which includes the constant factor C in a recurrence with the exponent is likely to produce better convergence and smaller constants. However, such a different analysis will not show that fewer than $O(\log\log n)$ iterations are necessary to find an $O(\sqrt{n})$-size separator. Even when the difficulties caused by constants are ignored and only the exponent is analyzed,

$O(\log\log n)$ iterations are still needed.

7 Conclusion

In this paper we have presented linear-processors NC algorithms for finding smaller and smaller cyclic separators in planar graphs which use more and more time. In $O(\log\log n)$ iterations the size of the separator is $O(\sqrt{n})$. The guaranteed rate of convergence can be traded off with the size of the constant on the $O(\sqrt{n})$-size separator.

A natural question is, what if there are more than n processors? With n^p processors, for $p \geq 1$, the exponent in Theorem 4 changes from $\left(1 - \frac{1}{\gamma M}\right)$ to $\left(1 - \frac{p}{\gamma M}\right)$. In other words, it converges much faster and with a smaller constant. Unfortunately, it still takes $O(\log\log n)$ iterations, and there still is no deterministic NC algorithm for finding an $O(\sqrt{n})$-size separator with $n^{M-\epsilon}$ processors for some constant $\epsilon > 0$.

Another natural question is, why can't the approach in this paper be made into an linear-processor NC algorithm? It's obvious that a great deal of recalculating of various divisions is being done. Why can't some of this information be saved? We think it can, but passing a separator through a division makes it difficult to control the constant factors associated with the number of regions and boundary vertices on the inside *and* outside of the cycle.

In this paper we have assumed that regions are composed of only triangulated faces. What happens when non-triangulated faces are allowed? Our techniques easily generalize to this situation and the resulting separator bounds discussed by Miller [18] and Gazit and Miller [9] immediately apply. In particular, though this is not the tightest bound, if d is the maximum face size then an $O(\sqrt{dn})$-size cyclic separator can be found. For our algorithms, if d is $O(n^\epsilon)$ for some constant $\epsilon > 0$ then we immediately have an NC algorithm for finding a cyclic separator which is with in a constant factor of the one produced by the best sequential algorithms.

One important consequence from the NC algorithms in this paper is that there now are *deterministic* NC algorithms which use fewer than n^2 processors for finding breadth-first-search trees in planar graphs. In particular, a corollary of Pan and Reif's work [21] is that given an algorithm for finding $O(n^{\epsilon})$-size separators, only $n^{3\epsilon}$ processors are necessary for BFS and single-source shortest paths. We show in a follow-up paper [24] to this one that only $n^{M\epsilon}$ processors are needed.

Acknowledgements

The authors would like to thank Greg Frederickson for fruitful discussions on this work and Angie Allen for proofreading our drafts.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] R. J. Anderson and G. L. Miller. Deterministic parallel list ranking. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures*, pages 81–90, 1988.

[3] R. Cole and U. Vishkin. Optimal parallel algorithms for expression tree evaluation and list ranking. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures*, pages 91–100, 1988.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987. NC integer matrix multiplication using $n^{2.376}$ processors.

[5] G. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, December 1987.

[6] G. Frederickson and R. Janardan. Efficient message routing in planar networks. *SIAM Journal on Computing*, 18:843–857, 1989.

[7] G. Frederickson and R. Janardan. Space-efficient message routing in c-decomposable networks. *SIAM Journal on Computing*, 19, 1990. To appear.

[8] H. Gazit. *Processor Efficient Parallel Graph Algorithms.* PhD thesis, University of Southern California, Los Angeles, CA, August 1988.

[9] H. Gazit and G. Miller. A parallel algorithm for finding a separator in planar graphs. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 238–251, 1987.

[10] H. Gazit and G. Miller. An improved parallel algorithm that computes the BFS numbering of a planar graph. *Information Processing Letters*, 28(2):61–65, June 1989.

[11] F. Harary. *Graph Theory.* Addison-Welsey, 1969.

[12] D. Johnson. Parallel algorithms for minimum cuts and maximum flows in planar networks. *Journal of the ACM*, 34(4):950–967, October 1987.

[13] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD 88/408, Computer Science Division (EECS), University of California, Berkeley, CA, March 1988. To appear in the *Handbook of Theoretical Computer Science* by North-Holland.

[14] P. Klein and J. Reif. An efficient parallel algorithm for planarity. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 465–477, 1986.

[15] S. R. Kosaraju and A. L. Delcher. Optimal parallel evaluation of tree-structured computations by raking. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures*, pages 101–110, 1988.

[16] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 36(2):177–189, April 1979.

[17] R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, August 1980.

[18] G. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, June 1986.

[19] G. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 112–117, 1989.

[20] G. Miller and J. Reif. Parallel tree contractions and its applications. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 478–489, 1985.

[21] V. Pan and J. Reif. Fast and efficient solution of path algebra problems. *Journal of Computer and System Sciences*, 38:494–510, 1989.

[22] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[23] G. Shannon. A linear processor algorithm for depth-first search in planar graphs. *Information Processing Letters*, 29(3):119–124, October 1988.

[24] G. Shannon and F. Wan. Improved processor bounds in NC algorithms for breadth-first search and single-source shortest paths on planar digraphs. Manuscript, 1990.

[25] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.

[26] R. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

[27] W. Tutte. *Graph Theory*, volume 21 of *Encyclopedia of Mathematics and its Applications.* Addison-Wesley, 1984.

[28] F. Wan. A linear-processor algorithm for finding small cycle separators on undirected planar graphs. Technical Report 308, Department of Computer Science, Indiana University, Bloomington, Indiana 47405, April 1990.

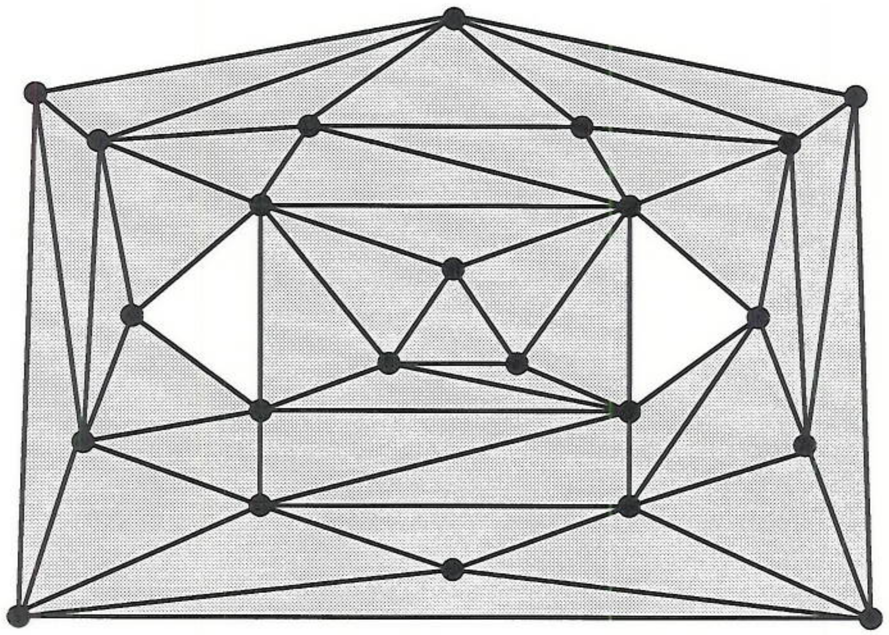


Figure 1. An embedded *region* of triangulated faces in light grey.

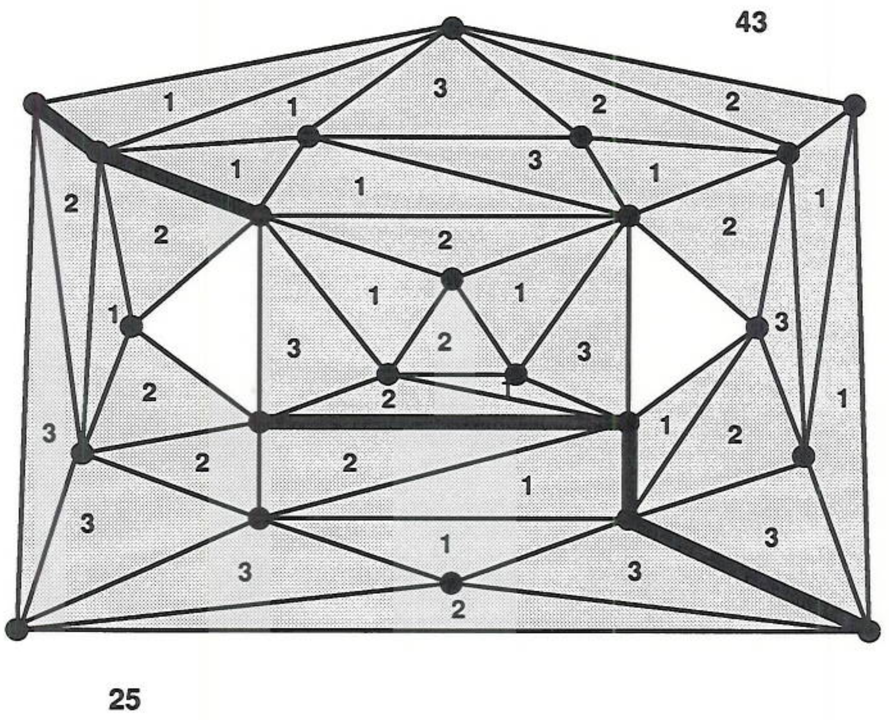


Figure 2. Weighted faces in a region and a *separator* (in heavy black) with respect to the face weights.

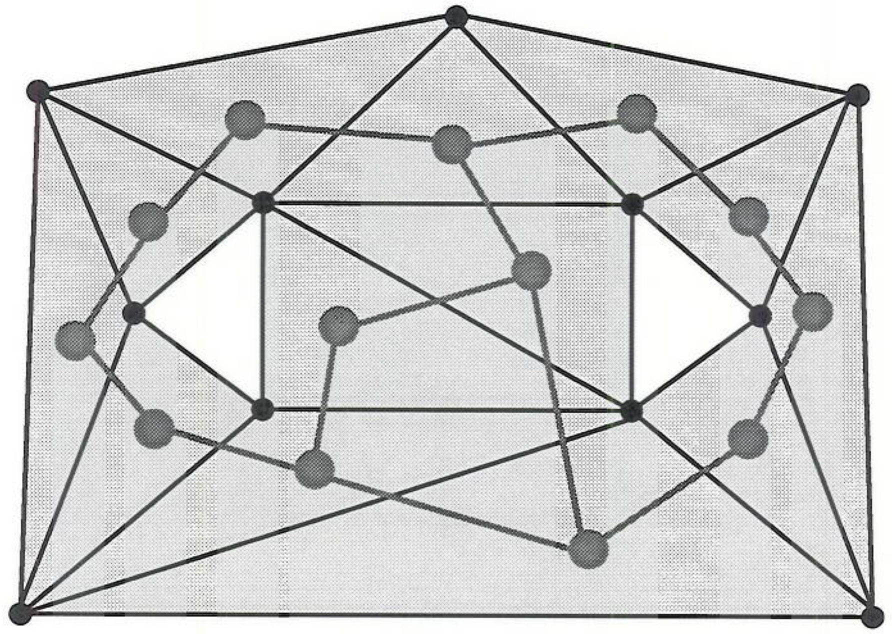


Figure 3. The *dual* of an embedded region. The dual's vertices and edges are in medium grey.

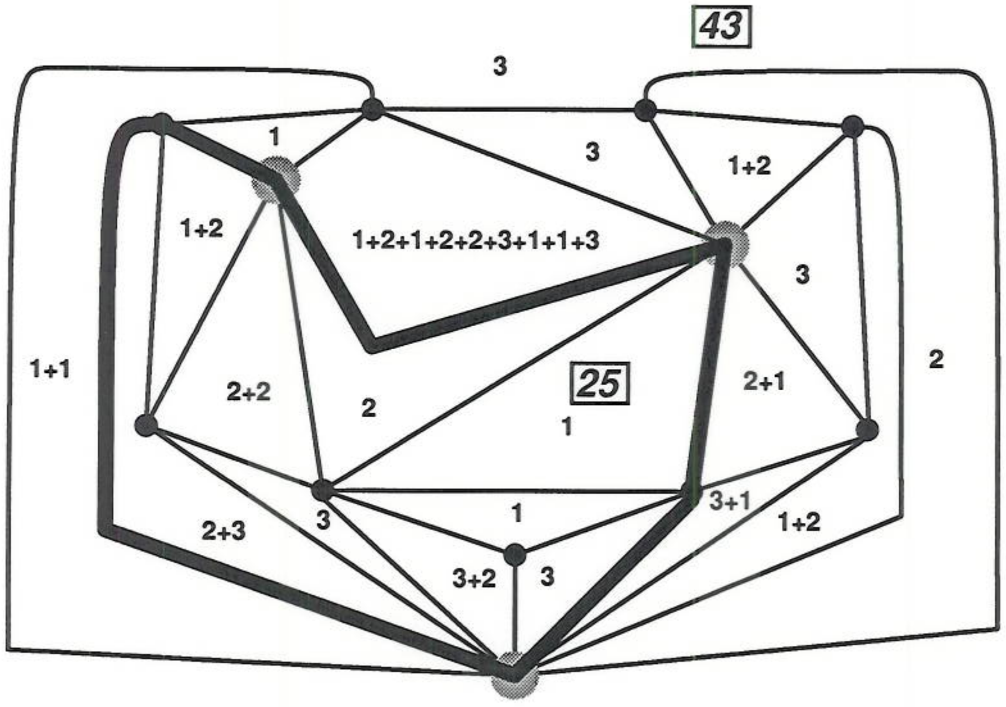


Figure 4. An application of the algorithm *Well-formed_Separator*. The above graph is after the boundary components have been collapsed (grey vertices) and duplicate edges have been removed. The heavy line is a cyclic separator in G' and implies the separator in Figure 2.

```
Algorithm:   Well-formed_Separator(c,ε)[R]
Input:       Region R of f triangulated faces.
Output:      Edge separator S for R with at most n^ε edges.

1.  Compute Boundary(R).
2.  Form R' by collapsing each component in Boundary(R) into one vertex.
3.  Find a spanning tree T of the dual of R'.
4.  Use T to find a 1- or 2-edge separator S for R' if it exists.
5.  Otherwise, use T to remove 1- and 2-edge separated components from R'.
6.  Apply the algorithm SEP(c,ε) to find a separator S in R'.
7.  Return S as the edge separator of R.
```

Figure 5. The algorithm *Well-formed_Separator*(c,ε) uses algorithm *Sep*(c,ε) to find a small edge separator in a region with triangular faces even though the boundary of the region may be very large.

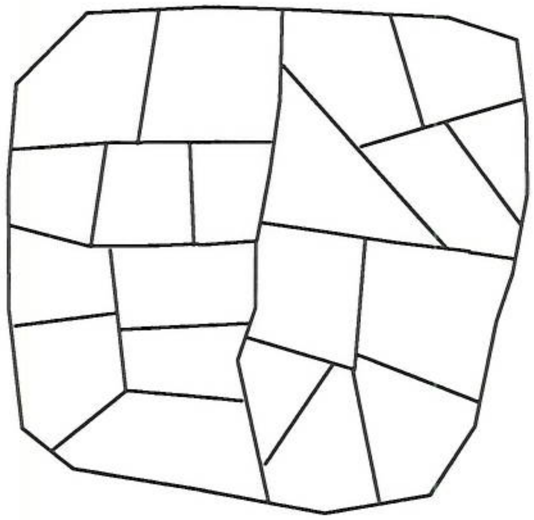
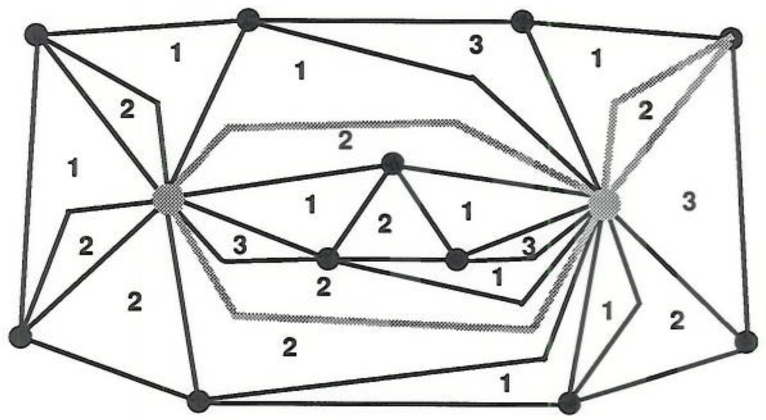
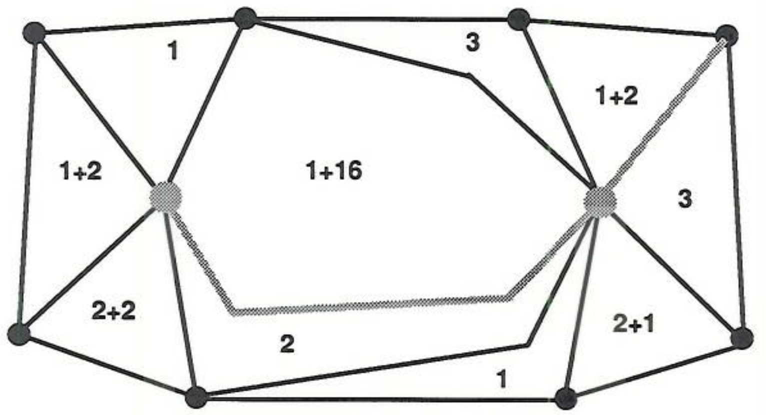


Figure 7. The structure of an r-division's skeleton.



(a)



(b)

Figure 6. Removing duplicate edges from a subgraph of G' from Figure 2. (a) Grey vertices are collapsed boundaries. The grey edges are examples of adjacent and non-adjacent duplicate edges. (b) The duplicate edges have been removed.

```
Algorithm:    Division(r,c,ε)[R]
Input:        Region R of f triangulated faces.
Output:       An (r,c,ε)-division of R.

0.   $P_0$ := {R}
     i := 0
     Boundary := Ø
1.   While ∃R ∈ $P_i$ such that Weight(R) > r Do
2.       For_Each R ∈ $P_i$ such that Weight(R) > r ParDo
3.           S := Well-formed_Separator(c,ε)[R]
4.           $P_{i+1}$ := $P_{i+1}$ ∪ { Inside(S,R) } ∪ { Outside(S,R) }
5.           Boundary := Boundary ∪ S
         End_For_Each
         i := i + 1
     End_While
6.   While ∃R ∈ $P_i$ such that Bound_Size(R) > $3 \cdot c \cdot r^{\varepsilon}$ Do
7.   For_Each R ∈ $P_i$ with $b_R$ boundary edges in Boundary PARDO
8.       For_Each R ∈ $P_i$ such that Bound_Size(R) > $3 \cdot c \cdot r^{\varepsilon}$ ParDo
9.           For_Each f ∈ R ParDo
                 Assign weight Boundary_Size(f)/Boundary_Size(R).
10.          S := Well-formed_Separator(c,ε)[R]
11.          $P_{i+1}$ := $P_{i+1}$ ∪ { Inside(S,R) } ∪ { Outside(S,R) }
12.          Boundary := Boundary ∪ S
         End_For_Each
         i := i + 1
     End_While
13.  Return $P_i$
```

Figure 8. The algorithm *Division*(c,ε) uses algorithm *Well–Formed_Separator*(c,ε) to find an (r,ε)-division of a region with triangular faces.

```
Algorithm:   Improved_Separator(c,ε)[G]
Input:       An n-vertex triangulated planar graph G.
Output:      A cyclic separator of size $c'n^{\varepsilon'}$ for G where $\varepsilon' < \varepsilon$.

1.  P := Divsion($n^{(1-1/M)}$,ε)[G]
2.  G′ := Skeleton(P,G)
3.  If G′ is biconnected then
4.      Find a cyclic separator S of G′ with Miller's algorithm.
5.  Otherwise, find a spanning tree T of the dual of G′.
6.  Find a balance point b of T.
7.  Let weight of b's subtree be $w_b$.
8.  If 1/3 ≤  ≤ 2/3 then
9.      Return b's boundary as S.
10. Otherwise, find the significant biconnected component C.
11. Find a cyclic separator S of C with Miller's algorithm after adjusting weights in C.
12. Return S.
```

Figure 9. The algorithm *Improved_Separator* uses algorithms *Sep*(c,ε) and *Division* to find a smaller separator for triangulated graph *G*.

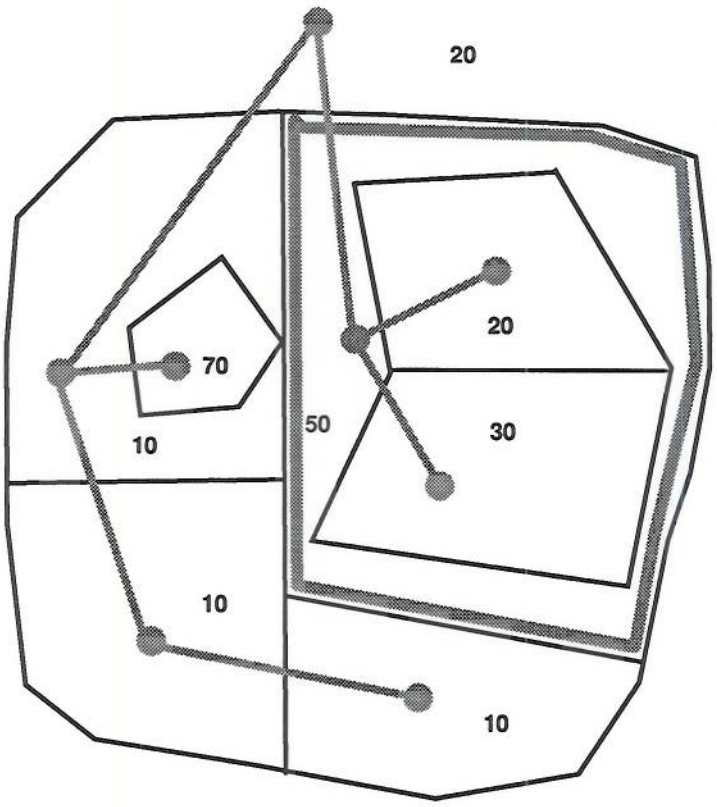
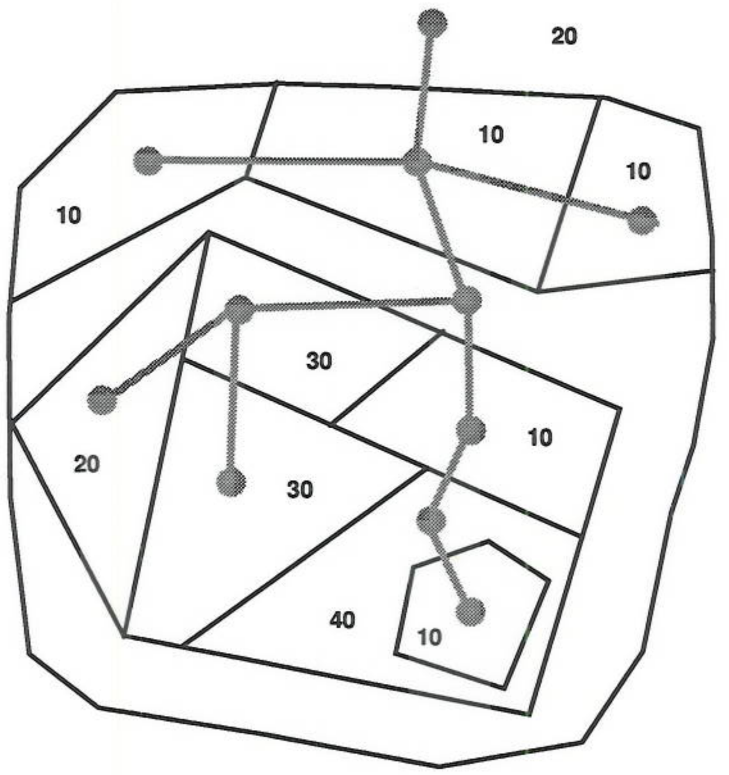
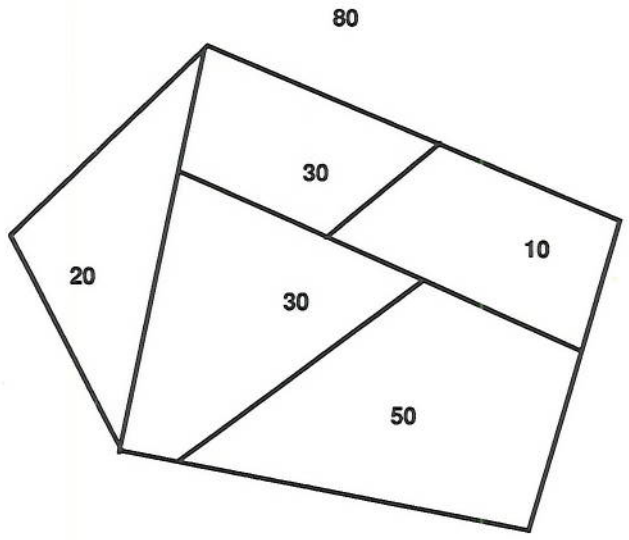


Figure 10. Finding a small separator in the skeleton of *G′* when it is connected. The grey spanning tree of the dual selects a balance point and implies the grey face boundary as a separating cycle.



(a)



(b)

Figure 11 . Finding a small separator in the skeleton of G' when it is not biconnected. The grey spanning tree of the dual (a) shows that the biconnected component in (b) will contain a separator after adjusting some faces' weights.