A Deterministic Parallel Algorithm for Finding a Separator in Planar Graphs [*]

Hillel Gazit [†] *Gary L. Miller* [‡]

Abstract

We present a deterministic parallel algorithm for finding a simple cycle separator in a planar graph. The size of the separator is $O(\sqrt{n})$ and it separates the graph so that the largest part contains at most $\frac{2}{3} \cdot n$ vertices. Our algorithm takes, in the PRAM EREW model, $T = O(\log^3(n))$ time and uses $P = O(n^{1+\epsilon})$ processors where n is the number of vertices, f is the number of faces and ϵ is any positive constant. The algorithm is based on a randomized solution from 1987.

Using a variation of our algorithm we can construct a simple cycle separator of size $O(d \cdot \sqrt{f})$ were d is maximum face size. The running time is the same, but the number of processors we need is $P = O(n + f^{1+\epsilon})$.

1 Introduction

A useful method for solving many problems is "divide-and-conquer". This method has been used to design sequential as well as parallel algorithms. In graph problems (and related problems), we usually need to find a separator in the graph in order to apply "divide and conquer".

A subset of vertices B is a **separator** if the remaining vertices can be partitioned into 2 sets A and C such that there are no arcs from A to C, and $|A|, |C| \leq \frac{2}{3} \cdot n$. The sets A, B, C form a partition of V.

*This work was supported in part by NSF grant DCR-8713489.

†Department of Computer Science,Duke University Durham, NC 27706

‡School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213-3890.

Finding a small separator can be used to solve problems such as layouts for VLSI [Lei80] [Val81], nested dissection [LRT79] in numerical analysis, and for efficient message routing [FJ86]. In particular Pan and Reif [PR85] showed how to solve the *BFS* problem given a family of small separators.

The simplest class of graphs with small separators is trees. Trees have separators of size 1. We will use this fact throughout this paper. Another important class of graphs with small separators is the planar graphs. A *planar graph* is a graph $G(V, E)$ that can be drawn on the plane, so that there are no arcs crossing. Many problems in VLSI layout and Numerical Analysis are posed for planar graphs.

Lipton and Tarjan [LT79] showed that for every planar graph we can find a separator B of size $\sqrt{8 \cdot n}$. Note that their result is *optimal* up to multiplicative constants in the worst case, since there exist planar graphs for which the smallest separator is of size $O(\sqrt{n})$. The $\sqrt{n} \times \sqrt{n}$ grid is an example.

There are many application of the parallel separator algorithm. In particular, the planar flow algorithms in [MN89] can be made deterministic with only a $\log n$ factor increase in the running time.

1.1 Previous Results

In 1979 Lipton and Tarjan [LT79] presented a sequential algorithm for finding a separator of size $\sqrt{8n}$ for planar graphs. In 1984 the Miller showed that a simple cycle separator, as defined in [Mil84], exists and can be found efficiently. In 1987 we [GM87] presented a randomized parallel algorithm with time complexity $O(\log^2(n))$ and processor complexity of $O(n^{1+\epsilon})$ for some constant ϵ greater than zero. The separator size is $O(\sqrt{n})$ but the constant depends on ϵ. A small ϵ increases the constant.

In this paper we present a deterministic version our parallel separator algorithm. The running time is the same as the running time of a deterministic algorithm for a maximal independent set. Right now the best result is $O(\log^3(n))$, and it was presented by Goldberg and Spencer [GS89]. If the maximal independent set algorithm will be improved in the future then our algorithm will be improved as well.

1.2 Preliminaries

2 Terminology and preliminaries

Throughout the paper let $G = (V, E)$ be an embedded planar graph where V is the *vertex set* and E is the *arc set*. An *arc* consists of two directed arc, a arc and its reflection. Let $R(e)$ be the permutation which takes an arc e to its reflection. The graphs considered may have multiple arcs, but every arc will have a distinct reflection. The tail of an arc e from vertex x to y is x, denoted by Tail(e).

A graph is said to be embedded in the plane if intuitively it is "draw" in the plane with no crossing arcs, an arc and it reflection are drawn on top of each other. This definition is not very algorithmic. So, all our algorithms assume that the graph is given by the following combinatorial definition of a planar embedding.

An embedding is a description of the cyclic orderings of the arcs radiating from each vertex. Formally, let $Sym(E)$ denote all permutations of the arcs of G.

Definition 2.1 *The permutation $\phi \in Sym(E)$ is an embedding of G if:*

1. *Tail(e)=Tail(ϕ(e)) for each dart $e \in E$.*

2. *ϕ restricted to the darts with tail at $v \in V$ is a cyclic permutation.*

Successive applications of ϕ will traverse the darts radiating from a vertex, in say, a clockwise order. On the other hand, the permutation $\phi^* = \phi \cdot R$ will traverse the darts forming the boundary of a face in counterclockwise order. We say that ϕ is a planar embedding if the number of faces f of the embedding satisfies Euler's formula:

$$f - e + v = 2$$

where e is the number of arcs and v is the number of vertices of the graph. A *region* will be the union of a collection of faces.

A *cycle* C is an ordered set of arcs $e_0, e_1, \ldots, e_k$ such that: for every $0 \leq i \leq k$, $head(e_i) = tail(e_{i+1}) \pmod{k+1}$. The cycle C is simple if the vertices between the arcs are distinct. Thus all cycles are *directed*. For planar embedding any simple cycle C has a well-defined interior **int(C)** (the faces, vertices, and arcs to the left of C) and a well-defined exterior **ext(C)** (the faces, arcs, and vertices to the right of C). Neither the int(C) nor the ext(C) contain the arcs or vertices of C.

A **weighted** planar embedded graph G is a triple $(G, \phi, \#)$ such that ϕ is a planar embedding of G and # is an assignment of weights to the vertices, arcs, and faces of G which sums to one such each face weight $\leq 2/3$.

A simple cycle C is a separator of a weighted embedded graph G if the weight of the interior and the weight of the exterior of C is at most $2/3$.

The breadth first search in our earlier work was performed in the face-incidence graph of G. The **face-incidence** graph has a vertex for each face of G and two faces are connected by an arc if they share a vertex in G. For technical reasons we shall use a slightly different graph which will simplify many of the definitions and algorithms.

Definition 2.2 *The* **face-vertex** *graph, denote $\mathcal{G}(V, E)$, is a bipartite embedded planar graph such that V is the set of the faces and vertices of G. A face F and a vertex v in G are connected by an edge if v is on the boundary of F.*

The distance between two face F and F' in $\mathcal{G}$ is the minimum number of vertices on any path in $\mathcal{G}$ from F to F'. A crucial definition throughout the paper is that of a neighborhood in a graph.

Definition 2.3 *If G is (possibly directed) graph and v is a vertex of G then a set N of vertices is a* **neighborhood** *of v if the distance from v to any vertex in N is at most the distance to any vertex not in N. A set N' is a* **subneighborhood** *of a neighborhood N of v if $N' \subseteq N$ and N' is a neighborhood of v. The set N is a* **k-neighborhood** *if N is neighborhood and the size of N is k.*

Note that a k-neighborhood need not be unique. We will restrict the neighborhoods in the face-vertex graph to consist of only faces ignoring the vertices.

2.0.1 The Algorithm

Let *Reduce-G*$(G(V,E),\ k)$ be an operation that reduces the planar graph $G(V,E)$ into another embedded planar graph $G'(V',E')$ such that G' is a subgraph of G and the number of faces in G' is $O(f/k)$. We describe this operation in detail later.

Our algorithm is given in Figure 1.

3 Basic Parallel Algorithms

We have several algorithms we will use later. The first algorithm finds the k-neighborhoods of each face as in [GM87]. This algorithm also gives us a *BFS* in faces within each of these neighborhoods. The second algorithm is the important one. The following is the crucial definition.

Definition 3.1 *A set MIS of neighborhoods* **dominates** *the set of k-neighborhoods MIS if:*

1. *The neighborhoods of MIS are face disjoint.*

In the following algorithm f_i represents the number of faces after iteration i, and $M(n)$ represent the complexity of algebraic matrix multiplication over the ring of integers.

```
1. $G_0(V_0, E_0) \leftarrow G(V, E); i \leftarrow 0;$
2. while $P < M(f_i)$ do
       $i \leftarrow i + 1; k \leftarrow \lfloor \sqrt[3]{\frac{P}{f_i}} \rfloor$
       $G_i(V_i, E_i) \leftarrow Reduce\text{-}G(G_{i-1}(V_{i-1}, E_{i-1}), k);$
3. Find a face $BFS$ of the faces $G_i$. Find a simple cycle separator in $G_i$ (using Miller's
   algorithm [Mil86]).
```

Figure 1: Outline of our algorithm

2. *Each neighborhood of MIS is a subneighborhood of some k-neighborhood.*

3. *Each k-neighborhood will have a nonempty intersection with some neighborhood in MIS.*

We find a dominating set of neighborhoods of size at most $2 \cdot \frac{f}{k}$. The third algorithm, using the results of the first and second algorithms, finds BFS of the face-vertex graph from at most $O(f/k)$ faces. Between each layer of faces we also get a layer in G of arcs. We divide each layer of arcs into simple cycles. Each cycle separates the graph between the inside and the outside. Therefore we can map (by homomorphism) the graph into a tree, were every cycle is mapped to an arc and the regions between the cycles are mapped to vertices. Using that tree, and the Tree Tour algorithm of Tarjan and Vishkin [TV85] we compute the weight inside and outside each cycle. We call the boundaries between faces that belongs to different "larger neighborhoods" the Voronoi Diagram of the graph. We give an algorithm that generates that Voronoi Diagram of the graph.

The algorithms are similar to those we use in our randomized paper. We make the appropriate changes so that we need not find a maximal independent set in the k-

neighborhood-intersection graph, and we will explain the changes in the next section.

3.1 Finding a Small Dominating Set of Neighborhoods

The idea is to find a dominanting set as defined in > Definition 3.1. Our estimate of the size will be factor of > 2 larger, but this will be good enough for our separator algorithm.

339,356d343

In this section we find a dominanting set as defined in Definition 3.1. In [GM87] we found instead a maximal face-disjoint set of k-neighborhoods, MIS. Recall that each k-neighborhood is a set of k faces. Thus, the number of neighborhoods in MIS is at most f/k. The approach was to find a maximal independent set in the neighborhood-intersection graph.

Definition 3.2 *The* **neighborhood-intersection** $\mathcal{G} = (V, E)$ *such that V is the set of faces of G, and two faces F, F' share an arc in E iff their k-neighborhoods $C(F), C(F')$ share a face.*

Observe that MIS is a maximal independent set in this graph. The problem is that this intersection graph may have $O(n^2)$ edges. Thus one can not at present use a determinist independent set algorithm directly. It will use n^2 processors. In the previous paper we went around the problem by using an implementation of Luby's randomized algorithm, [Lub86], in this paper we give a general scheme that can work with any efficient maximal independent set algorithm. The idea is to efficiently find a dominating set which is larger, by factor of 2, than the a maximal independent set. We change the rest of the algorithms in [GM87] so that they work with dominating sets of neighborhoods.

Definition 3.3 *The indegree of a face is the number of k-neighborhoods containing it.*

We say that a face has a high degree if it belongs to k^2 neighborhoods or more.

We change the graph $\mathcal{G}(V, E)$ by setting the neighborhood of every high degree face to the face itself (size one) and recomputing the intersections. Then we look for an independent set of neighborhoods in this new graph. we do it in the following way:

1. Include in the set all the modified neighborhoods which correspond to formally high degree faces.
2. Remove every neighborhood that contains a formally high degree face in its neighborhood from the list of neighborhoods.
3. Compute the reduced Neighborhood-Face graph.
4. Find a maximal independent set of neighborhoods in the remaining graph.

Lemma 3.4 *The independent set of neighborhoods as constructed above has size at most* $2 \cdot \frac{f}{k}$.

Proof: We have at most $\frac{f}{k}$ high degree faces, and every low degree face in the independent set has to have its own, unshared, k neighbors. Therefore at most $\frac{f}{k}$ low degree faces can be in the maximal independent set. □

Lemma 3.5 *The number of remaining pairs of intersecting neighborhoods in step 3. is at most* $f \cdot k^3$.

Proof: In the worst case we have $\frac{f}{k}$ faces with degree k^2; $\frac{f}{k} \cdot (k^2)^2 = f k^3$. □

Lemma 3.6 *The complexity of the maximal independent set procedure is* $O(\log^3(n))$ *using* $f \cdot k^3$ *processors.*

Proof: The proof follows from Lemma 3.5, by using Goldberg and Spencer [GS89] maximal independent set algorithm. □

4 Reduction Step

The union of faces and reduction of the graph is similar to what we did in our randomized separator paper. There are only two differences because of the new maximal independent set algorithm:

1. The upper bound for the independent set size is two times larger.

2. We need more processors.

Theorem 4.1 *There exists a parallel algorithm that given embedded weighted 2-connected planar graph G with f faces, such that each face is of size at most d and each face weight* $\leq 2/3$*, computes subgraph H that is 2-connected with at most* $6 \cdot f/k$ *faces, each of size at most* $d \cdot (4 + 5 \cdot \sqrt{k})$ *and each induced face weight* $\leq 2/3$*. This algorithm will use* $O(\log^3 n)$ *time and* $n \cdot k^3$ *processors in the EREW model.*

Proof: A full proof for a similar Lemma was given in [GM87]. □

5 Applications of Reduce-G

5.1 Stopping the recursion

We apply *Reduce-G* repeatedly till one of the following conditions hold:

1. The procedure *Reduce-G* returns a subgraph H which is a simple cycle separator.

2. The number of faces is small enough to compute a *BFS* of the graph H via matrix multiplication in $O(\log^2 n)$ using P processors.

5.2 Number of iterations

If we apply the procedure *Reduce-G* repeatedly, we can pick a larger and larger value for k. Therefore we will need only $O(\log(\log(f)))$ iterations to find a separator using $O(n)$

processors. We show that for $P = n^{1+\epsilon}$ a constant number of iteration is sufficient.

Define f_i to be the number of faces in the reduced graph after iteration i, and k_i to be the k used in iteration i.

In the first iteration $P = f \cdot k^3$ and $f_1 = 6 \cdot \frac{f}{k}$, but in the second iteration $P = f_1 \cdot {k_1}^3$, so

$$P = f \cdot k^3 = (\frac{6 \cdot f}{k}) \cdot k_1^3 \Longrightarrow k_1 = k^{\frac{4}{3}} / \sqrt[3]{6}$$

In the same way we can show that in the third iteration $k_2 = (k^{\frac{4}{3}})^{\frac{4}{3}} / \sqrt[3]{36} = k^{((\frac{4}{3})^2)} / \sqrt[3]{36}$ and in the i^{th} iteration $k_i = k^{((\frac{4}{3})^i)} / 6^{\frac{i}{3}}$. An upper bound for i can be obtained when $k_i \geq f$, since then we finish the algorithm. This implies $k_i = k^{(\frac{4}{3})^i} / 6^{\frac{i}{3}} < f$.

If we take $P = 2 \cdot f$ (that is, $k^3 = 2$), then an upper limit to the number of iterations is: $O(\log(\log(f)))$. If, on the other hand, we take $k = f^{\epsilon/3}$ where ϵ is some constant, then the number of iterations is a fixed constant i satisfying the equation $k^{\frac{4}{3}^i} / 6^{\frac{i}{3}} = f$. Note that i is constant that depends only on $\log(\frac{1}{\epsilon})$. A small ϵ means a large constant, and vice versa.

The separator size also increases also with the number of iterations. Using the result from our randomized separator paper we can conclude that the separator size will be about $(6 \cdot 5^2)^{i/3} \cdot 2 \cdot d\sqrt{f}$.

6 Open Problems

1. Is there an optimal polylogarithmic algorithm for the separator problem?

2. Is there a parallel BFS algorithm for planar graph that needs less than $n^{1.5}$ operations?

3. Is there a deterministic parallel isomorphism algorithm for planar graphs that needs less than $n^{\frac{7}{4}}$ operations?

References

[FJ86] Greg Fredrickson and Ravi Janardan. Separator-based strategies for efficient message routing. In *27th Annual Symposium on Foundations of Computer Science*, pages 428–437. IEEE, Oct 1986.

[GM87] Hillel Gazit and Gary L. Miller. A parallel algorithm for finding a separator in planar graphs. In *28th Annual Symposium on Foundations of Computer Science*, pages 238–248, Los Angeles, October 1987. IEEE.

[GS89] Mark Goldberg and Thomas Spencer. Constructing a maximal independet set in parallel. *SIAM J. Disc. Math.*, 2(3):322–328, August 1989.

[Lei80] C. Leiserson. Area-efficient graph layouts (for vlsi). In *21st Annual Symposium on Foundation of Computer Science*, pages 270–281. IEEE, Oct 1980.

[LRT79] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. on Numerical Analysis*, 16:346–358, 1979.

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. of Appl. Math.*, 36:177–189, April 1979.

[Lub86] M. Luby. A simple parallel algorithm fot the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, November 1986.

[Mil84] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 376–382, Washington,D.C., April 1984. ACM.