1988

A Linear-Processor Algorithm for Depth-First Search in Planar Graphs

Gregory E. Shannon

Report Number:
88-737

A LINEAR-PROCESSOR ALGORITHM FOR DEPTH-FIRST SEARCH IN PLANAR GRAPHS

Gregory E. Shannon

CSD-TR-737
January 1988

A Linear-Processor Algorithm for Depth-First Search in Planar Graphs*

Gregory E. Shannon†
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

Abstract

We present an n-processor and $O(\log^2 n)$-time parallel RAM algorithm for finding a depth-first-search tree in an n-vertex planar graph. The algorithm is based on a new n-processor algorithm for finding a cyclic separator in a planar graph and Smith's original parallel depth-first-search algorithm for planar graphs [Smi86].

Keywords: Analysis of algorithms, parallel algorithm, planar graph, depth-first search, cyclic separator.

*A preliminary, abstracted version of these results appeared as part of [GPS87].

†Supported in part by Hewlett-Packard's Faculty Development Program, NSF grant DCR-8320124, and ONR contract N00014-86-K-0689.

1. Introduction

Efficient sequential algorithms for depth-first search (DFS) in a graph [Tar72] have proven very valuable in the area of sequential algorithms. The use of DFS in parallel algorithms, however, has been limited because the previously known algorithms are randomized for general graphs [AA87] and/or use a large number of processors for planar graphs, at least n^2 [Smi86,Hag87]. Using a new linear-processor algorithm for finding a cyclic separator in planar graphs, which we present here, and Smith's DFS algorithm in [Smi86], we produce a fast linear-processor DFS algorithm for planar graphs on the parallel RAM (PRAM) model of computation. This helps make DFS a more useful tool in designing parallel algorithms.

Smith presented in [Smi86] a parallel algorithm for constructing a depth-first-search (DFS) tree in a connected undirected planar graph. His algorithm used $O(\log^3 n)$ time and n^4 processors for an n-vertex graph. In light of some recent results in parallel algorithms, the only step in Smith's algorithm which still uses more than a linear number of processors is the one to find a cyclic separator of a planar graph. A *cyclic separator* of a graph is a cycle such that neither its interior nor its exterior contains more than two-thirds of the graph's vertices. Smith's own separator algorithm used n^4 processors. In this paper, we present a cyclic separator algorithm which uses n processors and $O(\log n)$ time on a concurrent-read concurrent-write (CRCW) PRAM. Our algorithm is based on using arbitrary spanning trees of the dual graph to impose structure so that the problem can be efficiently simplified. Another parallel cyclic separator algorithm for planar graphs by Miller [Mil86] uses as many processors as parallel matrix multiplication, $\Omega(n^2)$, and $O(\log n)$ time, but the separator has only $O(\sqrt{n})$ edges. With our new algorithm, Smith's algorithm now uses only n processors and $O(\log^2 n)$ time on the CRCW PRAM or $O(\log^2 \log^* n)$ time on the exclusive-read exclusive-write (EREW) PRAM. All of our algorithms use linear space.

Our graph terminology and notation is based on [BM76] and [Har69]. Since there

is a linear-processor and poly-log algorithm for embedding a planar graph in the plane [KR86], we consider finding an embedding a separate issue and assume that the input graph is already embedded. The input graph is represented with linked lists as discussed in [TV85]; this includes two doubly-linked directed edges for each undirected edge. The embedding is represented by doubly-linked ordered edge lists as in [LT79]. Since an n-vertex planar graph has only $O(n)$ edges this representation uses only $O(n)$ space. We design our algorithms assuming that one processor is associated with each vertex and edge in the graph. Therefore, processor assignment is not an issue here. In the CRCW-PRAM model, concurrent writes are resolved arbitrarily. The PRAM models of computation we use are discussed in more detail in [BH85].

In the next section we present and analyze our fast linear-processor separator algorithm. In section 3 we show how this algorithm, combined with other recently improved algorithms, reduces the processor and time complexity of Smith's DFS algorithm.

2. Finding a Cyclic Separator

In this section we present a new linear processor algorithm for finding a cyclic separator of an undirected, biconnected, and embedded planar graph.

Our algorithm actually solves the more general problem of finding a cyclic separator of a face weighted plane graph. A cycle separates a plane graph with weighted faces if the sum of the faces in the interior of the cycle is at most $\frac{2}{3}$ and likewise for the exterior of the cycle. To solve the problem of finding a vertex separator, we add appropriate weights (discussed below) to G's faces so that a cyclic separator of G's weighted faces also separates G's vertices.

To find a separator of G's weighted faces, we either find a face whose boundary can be used as a separator or, if there is no such face, find faces which can be merged into a superface whose boundary is a separator. Note that every face in a biconnected

plane graph is a cycle [Tut84]. Naive merging of adjacent faces does not work because the boundary of the resulting superface might not be a simple cycle [Smi86,Tut84]. By merging only faces associated with a subtree of a spanning tree of the dual D of G, we can guarantee that the boundary of each superface produced is a simple cycle.

Given an n-vertex undirected biconnected plane graph G, the 4-pass algorithm below returns a subset of G's edges which is a cyclic separator of G's vertices.

Pass 1. Assign weights to G's faces as follows. Allocate weight $1/n$ to each vertex. Each vertex then independently adds its weight to a face adjacent to an edge incident to it. Clearly, a separator of G's weighted faces also separates G's vertices. If there is a face with a weight at of least $\frac{1}{3}$, then return its boundary as the separator.

The algorithm now proceeds until some superface's weight in G is *heavy* (between $\frac{1}{3}$ and $\frac{2}{3}$). The boundary of such a heavy face is a separator of G's weighted faces (and vertices) since a superface's weight is the sum of the weights of its component faces.

Pass 2. If no face is heavy in G, then find an arbitrary rooted spanning tree T of the dual D of G. Such a tree exists since D is connected if G is connected [Tut84]. Define F to be the superface formed by merging the faces associated with the subtree rooted at f in T. The boundary of F is simple since F and $T - F$ are both connected [Tut84,Smi86].

Find a *critical vertex* f_c in T such that the weight of F_c is at least $\frac{1}{3}$ and the weight of each superface F_i is less than $\frac{1}{3}$ where the $\{f_i\}$ are the critical vertex's children in T. At least one critical vertex exists if the sum of the weights in G is at least $\frac{1}{3}$.

Consider a new dual D' of G's superfaces $\{F_i\}$, F_r (the merge of the faces remaining in $T - F_c$), and the face C corresponding to f_c. These faces form a biconnected plane graph since each face's boundary is a simple cycle [Tut84].

Each face associated with D' has a weight of at most $\frac{2}{3}$. By the definition of f_c, F_r and the $\{F_i\}$ are not heavy. C is not heavy since G has no heavy faces. Return the boundary of F_r as the separator if F_r is heavy. Since G has no heavy faces and

by the definition of f_c, only F_r could be heavy.

Pass 3. If no superface associated with D' is heavy, find a spanning tree T' of D'. Its root must be C and have only one child. Such a tree exists since D' is biconnected iff the underlying plane graph is biconnected [Tut84]. Find a critical vertex f'_c in T'; its associated face in D' is C'. The structure of T' guarantees that C' is different from C. Define and construct $\{f'_i\}$, $\{F'_i\}$, F'_r, and D'' analogously as in pass 2. By construction, D'' has the same properties as D'. Return the boundary of F'_r as the separator if F'_r is heavy.

Pass 4. If no superface associated with D'' is heavy, naively and accumulatively merge the $\{F'_i\}$ superfaces one at a time with C' until one of the generated superfaces H is heavy. The boundary of H is then a separator of G.

Why is such an H guaranteed to exist? We first need to show that each superface generated has a simple boundary. Let $H_0 = C'$, and let H_{i+1} be the merge of H_i and F'_i, for i up to $|\{F'_i\}|$. As discussed in pass 2 for the boundaries of the faces in D', if the component faces in H_i can be associated with a subtree of a spanning tree of D'' then H_i has a simple boundary. Given the way D'' (D') was constructed, it is easy to see that each vertex in D'' (D') is adjacent to the vertices associated with C' (C). Since D'' is formed from D' by contracting edges (merging faces), C is not equal to C', and C is a component face in F'_r, each vertex in D'' is adjacent to the vertex associated with F'_r and C. Therefore, with respect to vertices associated with the faces, for any subset X of $\{F'_i\}$, a tree exists where all faces in X are in a subtree rooted at F'_c, F'_c is a child of F'_r, and all faces in $\{F'_i\} - X$ are children of F'_r. Therefore, all H_i have simple cycles for boundaries.

Now we need to show that one of the generated superfaces is heavy. Since no face in D'' is heavy, some superface H_i has a weight of less than $\frac{1}{3}$ and some superface H_{i+1} has a weight of at least $\frac{1}{3}$ but no more than $\frac{2}{3}$, assuming the total weight on the faces is at least $\frac{1}{3}$.

Therefore, the 4 passes above correctly find a cyclic separator in G.

We now describe and analyze below a CRCW-PRAM implementation of the above algorithm for an n-vertex biconnected plane graph $G = (V, E)$. Since an n-vertex planar graph has $O(n)$ edges, each sub-algorithm discussed below uses at most n processors and $O(n)$ space, and G has a linear-space linked-list representation (as discussed at the end of the introduction). Therefore, the implementation uses only $O(n)$ processors and space, and we analyze only the time complexity of the implementation.

Pass 1. Each processor arbitrarily selects an incident edge to assign its weight of $1/n$ to. The weight of each face is found using the standard $O(\log n)$-time recursive doubling algorithm to sum the weights assigned to the edges in the face's boundary.

Pass 2. Use the spanning tree algorithm of Shiloach and Vishkin [SV81] to find T. Viewing T as an expression tree which sums the faces' weights, use the $O(\log n)$-time tree-expression evaluation algorithm of Miller and Reif [MR85] to compute the weight in each of T's subtrees.

Using concurrent writes, each vertex marks its parent in T if the vertex's subtree's weight is greater than $\frac{1}{3}$. Each unmarked vertex with a subtree weight of at least $\frac{1}{3}$ then writes to a global location to decide the critical vertex f_c in T. Both of these steps take constant time.

To construct the faces associated with D', we must determine which edges remain after merging faces. Let f_{xy} be a face with edge (x, y) in its boundary. For each face f in G, let $super(f)$ be the superface associated with D' in which f is a component face. Compute *super* for each face using Miller and Reif's $O(\log n)$-time tree-contraction algorithm [MR85]. An edge (x, y) is in the boundary of a superface associated with D' iff $super(f_{xy}) \neq super(f_{yx})$. Given the representation of G and the values of *super*, this inequality can be tested in constant time by each pair of directed edges representing an edge in G. Constructing the lists of superface edges and D' then takes only $O(\log n)$ time using recursive doubling on G's edge lists.

Pass 3. The root of T' and its only child are easily determined with one processor.

The rest of the pass is implemented as pass 2 is.

Pass 4. Since the superfaces associated with D'' are linked together as a list, a parallel linked list prefix sums algorithm based on recursive doubling and using $O(\log n)$ time can easily find an appropriated subset of the $\{F'_c\}$ faces to merge with C' to form a superface whose boundary is a separator. Use the techniques from pass 2 to determine which edges are in the separator.

Therefore, the cumulative time complexity of this algorithm is $O(\log n)$, and we have the following theorem.

Theorem. *Given an n-vertex undirected biconnected plane graph G, the algorithm presented above correctly returns a subset of G's edges which is a cyclic separator of G's vertices. The implementation of the algorithm discussed above uses n processors, $O(\log n)$ time, and $O(n)$ space.*

3. The Depth-First-Search Algorithm

Since finding a cyclic separator was a key subproblem in Smith's DFS algorithm, combining Smith's algorithm, our new separator algorithm, and other recently improved parallel algorithms implies a better DFS algorithm as stated in the theorem below.

Theorem. *A DFS tree can be found using n processors and $O(\log n)$ time on the CRCW PRAM or $O(\log n \log^* n)$ time on the EREW PRAM.*

A brief overview follows of Smith's algorithm for constructing a DFS tree T in G. Without loss of generality, assume G is biconnected and embedded. Assume T is to have root x. Find a cyclic separator S of G. Find a path P from S to x with no vertex in S in P's interior. Select an edge e in S next to where P meets S. Update T by adding in $P \cup S - \{e\}$. Compute the distance of each vertex in T from x using edges only in T. For each biconnected component B' in $G - T$, determine which vertex x' has an edge f connecting x' to the lowest numbered vertex in T of all the vertices in B'. Add f to T and recurse on each B' and x'.

The above algorithm has $O(\log n)$ levels of recursion, and the time needed at each level depends on the time complexity of the algorithms for cyclic separator, biconnected components, spanning tree, tree traversal and labeling, and linked list traversal and labeling. By using our separator algorithm and more efficient algorithms than Smith used for biconnected components and spanning trees [TV85], each level of the algorithm will use only $O(\log n)$ time and n processors. The total algorithm uses then $O(\log^2 n)$ time and n processors to find a DFS tree in a planar graph.

To achieve the claimed results on the EREW PRAM we rely on a recent $O(\log n \log^* n)$ algorithm for finding connected components in planar graphs [LP86]. Biconnected components and spanning trees can be found in the same time using an n-processor and $O(\log n)$-time reduction to connected components due to Tarjan and Vishkin [TV85]. This leaves only a constant number of time steps where concurrent writes were used in our CRCW PRAM algorithm. All of these can be resolved with recursive doubling on the underlying linked-list data structured in $O(\log n)$ time. Therefore, the time bound claimed for the EREW PRAM holds.

Acknowledgements

I would like to thank Greg Frederickson and Susan Rodger for reviewing early drafts of this paper.

References

[AA87] A. Aggarwal and R. Anderson. A random NC algorithm for depth first search. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 325–334, 1987.

[BH85] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[BM76] J. Bondy and U. Murty. *Graph Theory with Applications.* North-Holland, 1976.

[GPS87] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 315–324, May 1987.

[Hag87] T. Hagerup. *Planar DFS in $O(\log n)$ Parallel Time.* Technical Report 1987-18, Informatik, Universitaet des Saarlandes, Saarbruecken, West Germany, October 1987.

[Har69] F. Harary. *Graph Theory.* Addison-Welsey, 1969.

[KR86] 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.

[LP86] C. Leiserson and C. Phillips. A fast parallel algorithm for region labeling. October 1986. Extended abstract.

[LT79] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 36(2):177–189, April 1979.

[Mil86] G. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, June 1986.

[MR85] 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.

[Smi86] J. Smith. Parallel algorithms for depth-first searches I. planar graphs. *SIAM Journal on Computing*, 15(3):814–830, August 1986.

[SV81] Y. Shiloach and U. Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *Journal of Algorithms*, 2:88–102, 1981.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[Tut84] W. Tutte. *Graph Theory.* Volume 21 of *Encyclopedia of Mathematics and its Applications*, Addison-Wesley, 1984.

[TV85] R. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, November 1985.