SRC TR 86-86

Parallel Algorithms for Planar Graph Isomorphism and Related Problems

by

Joseph Ja'Ja' and S. Rao Kosaraju

Parallel Algorithms for Planar Graph Isomorphism and Related Problems

Joseph Ja'Ja'[†]
Department of Electrical Engineering
Institute For Advanced Computer Studies
and
Systems Research Center
University of Maryland
College Park, Maryland 20742

and

S. Rao Kosaraju[*]
Department of Electrical Engineering and Computer Science
The Johns Hopkins University
Baltimore, Maryland 21218

[†]Supported by the U.S.Army Research Office, Contract No. DAAG-82-k-0110, NSA Contract No. MDA-904-85H-0015, NSF Grants No. DCR-86-00378, and No. CRD-85-00108.

[*] Supported by NSF Grant #DCR-8506361

1. Introduction

Let $G = (V,E)$ be a planar graph embedded in the plane. We assume that the embedding is specified by giving an orientation to G [11]. Planar graph isomorphism can be reduced to finding the triconnected components of the two graphs involved, partitioning these components into isomorphism equivalence classes and testing the isomorphism of the correponding 3-connected trees [8]. Since fast parallel algorithms for tree isomorphism are known ([11],[12]), we concentrate here on the problems of identifying the triconnected components of planar graphs and on testing the isomorphism of triconnected graphs. The latter can be viewed as a special case of the general coarsest partitioning problem [1] which will also be considered in this paper.

We present several new efficient parallel algorithms for the above problems. The complexity of these algorithms will be examined in the context of two models of parallel computation: the concurrent-read exclusive-write parallel random access machine (PRAM), and the two dimensional array of processors. We assume in the rest of the paper that the reader is familiar with the basic parallel techniques for both of these models. In addition, some familiarity with graph theory will also be assumed.

For the array model, we have obtained optimal algorithms for several nontrivial problems related to planar graph isomorphism and which are important on their own. Suppose that the array of processors consists of an $\sqrt{n} \times \sqrt{n}$ grid of processors where the input is of size n. Then an $O(\sqrt{n})$ time algorithm for finding a good *separating cycle* was developed. This algorithm can be used to find a depth-first spanning tree of a planar graph in

$O(\sqrt{n})$ time. The algorithm can also be implemented on a PRAM with O(n) processors and $O(log^3 n)$ time, which is an improvement over the $O(n^4)$ processors and $O(log^3 n)$ time of Smith [13] (even excluding the part that computed the planar embeddings). Finding the triconnected components of a planar graph can also be done in $O(\sqrt{n})$ time on the array which translates into an $O(log^3 n)$ time algorithm with $O(n)$ processors on the PRAM. Our algorithm is based on a new divide-and-conquer strategy that uses several nontrivial facts shown to hold for the triconnected components of planar biconnected graphs. Independently Miller and Ramachandran [10] have developed a fast parallel algorithm for finding the triconnected components of an arbitrary graph with $O(n)$ processors. However their algorithm does not seem to be implementable on the mesh.

Another interesting implementation we developed is a fast array algorithm for the single function *coarsest partitioning* problem [1]. None of the known sequential algorithms seem to translate into $O(\sqrt{n})$ mesh algorithm. Our $O(\sqrt{n})$ implementation makes use of several nontrivial constructions. For example, given a sequence $a_1, a_2, \ldots, a_n$, the mesh can find the smallest i such that there exists a j satisfying $a_1 a_2 \cdots a_n = (a_1 a_2 \cdots a_i)^j$ in $O(\sqrt{n})$ time.

We also present an isomorphism algorithm for triconnected planar graphs which runs in time $O(log^2 n)$ time with $O(n^2)$ processors. Notice that the best known previous NC-algorithm uses $O(n^4)$ processors [11]. Therefore planar graph isomorphism can be done within the same time and processor bounds. On the other hand, we show that the general coarsest par-

titioning problem is in NC. This algorithm can be used to solve the problem of minimizing the states of a finite state automaton and the equivalence problem of any two finite state automata.

2. Finding a Separating Cycle

An important strategy for solving planar graph problems is based on a divide-and-conquer approach that results from identifying a good separator. Our method for determining the triconnected components of a planar graph depends on finding a *separating cycle* C that, unlike other known separator cycles, separates the *edges* of the planar graph in such a way that either C is a face with a large number of edges or the number of edges inside or outside C is a constant fraction of the total number of edges. The algorithm is given below. For a given cycle C, Int(C) and Ext(C) denote the sets of edges in the interior and exterior of C respectively.

Algorithm Cycle

Input: A biconnected planar graph G=(V,E) given by {(f,e)| e is an edge of face f}, for all interior faces f.

Output: A set of edges C that form a cycle such that either

(a) C is a *face* such that $|C| \geq \frac{m}{8}$

or

(b) $|C| + |Int(C)| \leq \frac{7m}{8}$ and $|Ext(C)| \leq \frac{7m}{8}$,

where |E|=m.

1) If there exists a face f such that $|f| \geq \frac{m}{8}$, then set C=f and exit. Else

go to step 2.

2) Construct a modified dual G^* of G.
   a) sort according to (e,f), where e is an edge of the face f.
   b) create edge (f_1, f_2) for every pair (e, f_1) and (e, f_2).
   c) sort edges (f_1, f_2) and delete duplicates.

3) Find a spanning tree T of G^* and make it directed with an arbitrary root f_r. For each node f in T, let size(f) be the number of edges in the face f.

4) For each node f in T compute:

$$SIZE(f) = \sum_{f_d \text{ is a descendant of } f} size(f_d)$$

5) Identify a node v of T such that

$$\frac{m}{4} \leq size(v) + \sum_{i=1}^{s} SIZE(v_i) \leq \frac{3m}{4}$$

where the set $\{v_i \mid 1 \leq i \leq s\}$ is a subset of the children of v.

6) Set C= the exclusive-OR of the edges in v and the edges in $\{v_i \mid 1 \leq i \leq s\}$ and their descendants.

We are ready for our first theorem.

Theorem1 : Given a biconnected planar graph, **Algorithm Cycle** correctly finds a separating cycle satisfying the conditions stated above. Moreover, the algorithm can be implemented to run in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh.

Proof: Clearly if the condition in step 1 holds then we are done. Hence assume that $|f| < \frac{m}{8}$, for all faces f.

Claim: v of step 5 exists.

Proof of Claim: Start from the root f_r of T and go down the tree along a path containing nodes such that $SIZE(v) > \frac{3m}{4}$. Let u be such that

$SIZE(u) > \frac{3m}{4}$ and for all sons u_i's, $1 \leq i \leq t$, $SIZE(u_i) \leq \frac{3m}{4}$. If there exists a j such that $SIZE(u_j) \geq \frac{m}{4}$, then u_j with all its children satisfy the conditions of step 5. Else assume that $SIZE(u_i) < \frac{m}{4}$, for all i. Notice that

$$SIZE(u) = size(u) + \sum_{i=1}^{t} SIZE(u_i) > \frac{3m}{4}$$

$$size(u) < \frac{m}{8} \text{ and } SIZE(u_i) < \frac{m}{4} \text{ for } all \ i.$$

Clearly, $v = u$ satisfies the condition stated in 5).

We now show that the cycle C produced at step 6) satisfies the desired properties. It is clear that C satisfies

$$\frac{m}{4} \leq |C| + 2|Int(C)| \leq \frac{3m}{4}$$

Using the above inequality, a simple argument will show that C is a separating cycle.

Steps 1-3 can be easily implemented on the mesh in $O(\sqrt{n})$ time. Step 4 can be done by the Euler tour technique. Wrap a chain around the tree and assign weights appropriately. Compute the rank of each node ([2]). Step 5 can be done as follows. Sort the pairs (u_i, u) according to u, where u_i is the son of u. Identify a node u such that $SIZE(u) > \frac{3m}{4}$ and $SIZE(u_i) \leq \frac{3m}{4}$. From this it is easy to complete step 5. ●

3. Depth-First Spanning Tree

Smith has presented an algorithm in [13] to determine a depth-first spanning tree of a planar graph in $O(\log^3 n)$ using $O(n^4)$ processors even excluding the part that finds the planar embedding. Next we present an algorithm essentially deduced from Smith's that runs in $O(\sqrt{n})$ time on the mesh. This algorithm can be implemented on a PRAM in $O(\log^3 n)$ time and O(n) processors. A brief sketch of the algorithm is given below.

Algorithm Depth-First Search

Input: A planar connected graph given by {(f,e)| e belongs to face f} and a vertex v to be the root of the Depth-First Spanning (DFS) tree.

Output: A set of edges forming a DFS tree T of G rooted at v.

1. Construct the block-cutpoint graph. All the bridges[1] belong to T. Root this tree at v if v is a cutpoint or at the vertex b(v) corresponding to the biconnected component containing v.
For each vertex i of T corresponding to a biconnected component B_i, find a cutvertex v_i of B_i that is closest to v (or b(v)).

2. For each (B_i, v_i), do the following:

   (a) Use Algorithm Cycle to find a separating cycle C_i of B_i.
   (b) Find a path P_i from v_i to C_i.
   Remark: This can be obtained by finding a spanning tree T_i of B_i with root v_i. Find closest vertex w of C_i to v_i. P_i is the path from v_i to w in T_i.
   (c) Construct path $Q_i = P_i \bigcup C_i$-an edge e in C_i with one endpoint in P_i. Root this path at v_i. Include Q_i in T. Find the depths of the vertices in Q_i (needed later).
   (d) Find the connected pieces of $B_i - Q_i$.
   (e) From each connected piece, add an edge to T incident on Q_i and

[1] In this case, a *bridge* is an edge whose removal disconnects the graph.

furthest from v_i.
(f) Remove all edges between the connected pieces and Q_i and apply the algorithm recursively to each resulting connected component.

It is not hard to establish the above algorihtm's correctness and complexity. We next address the problem of finding the triconnected components of a planar graph.

4. Determining the Triconnected Components

There are several ways of defining the triconnected components of a graph, all of which are more or less equivalent. We will essentially adopt the definitions given in [6].

Let H=(S,T) be a subgraph of a given biconnected multigraph G=(V,E). Suppose that E is partitioned into a set $\{E_1,E_2,...,E_k\}$ such that two edges e and g belong to the same partition if, and only if, e and g are on a path in which none of its internal vertices belong to S. Then the sets $E_1,E_2,...,E_k$ are called *the bridges* of G relative to H. A pair of vertices {u,v} form a *separation pair* if there are at least two bridges of G with respect to {u,v} except in the following two cases: (i) G has exactly two bridges and one of them consists of a single edge, (ii) there are exactly three bridges each of which consists of a single edge.

A biconnected multigraph is *triconnected* if it has no separation pairs. Otherwise decompose the edges into two sets E' and E'' such that a bridge is contained in only one of them and $|E'| \geq 2$, $|E''| \geq 2$. Augment each of the

corresponding subgraphs G_1 and G_2 with *the virtual edge* {u,v}. G_1 and G_2 are called *split graphs* with respect to {u,v}. We can continue the splitting process until no more splits are possible. The subgraphs obtained are called the *split components* of G. If the triple bonds and the triangles are merged to bonds and polygons, the resulting components are called *the triconnected components* of G. It is shown in [6] that these components are unique.

Since a graph may have "too many" separation pairs, we define a *minimal complete* set of separation pairs to be a set S of separation pairs such that :

(i) If {u,v} is a separation pair of G, then either {u,v} is in S or u and v belong to two distinct bridges relative to some separation pair in S.

(ii) Given any two separation pairs {u,v} and {r,s} in S, then r and s belong to the same bridge relative to {u,v}.

It is clear that such a set of separation pairs exists. Notice that in general many such sets may exist. Our algorithm for identifying the triconnected components will start by finding a complete set of separation pairs.

Identifying the separation pairs of a planar graph is a key step in determining the triconnected components. Our algorithm is based on several facts that hold only for planar biconnected graphs.

Lemma1: Let {u,v} be a separation pair of a planar biconnected graph G=(V,E). Then u and v must belong to a face f for any planar embedding of G.

Proof: Suppose u belongs to the faces $f_1, f_2, \ldots, f_s$ and suppose that v belongs to $f_1', f_2', \ldots, f_t'$, where all these faces are distinct. The faces form a plane mesh. Since {u,v} is a separation pair, there must exist two edges e_1 and e_2 incident on u such that every path containing e_1 and e_2 must contain either u or v. Clearly, this is not the case here, and hence there is a face containing u and v. ●

Note that Lemma1 does not hold if G is not biconnected.

Theorem2: Suppose {u,v} is not a multiple edge. Then {u,v} is a separation pair if and only if either (i) or (ii) holds.

(i) u and v belong to three or more distinct faces, one of which could be the exterior face.

(ii) {u,v} is not an edge, u and v belong to two faces, one of which could be the exterior face.

Proof: By Lemma1 we know that u and v must belong to a face f.

Let {u,v} be a separation pair. Let C_f be the cycle determined by f. Consider the bridges of G relative to C_f. Two cases arise:

(a) u and v are the only vertices of attachment of a bridge B. In this case, it is clear that (i) holds.

(b) Let p_1 and p_2 be the segments of C_f determined by u and v. Clearly neither p_1 nor p_2 consists of a single edge. All bridges must have their vertices of attachment either in p_1 or in p_2. Clearly (ii) holds in this case.

Now suppose that (i) holds, one of the bridges must have u and v as the only vertices of attachment. Hence {u,v} is a separation pair.

Suppose (ii) holds. If the other face turns out to be the exterior face, then {u,v} is clearly a separation pair. Otherwise, let $\bar{f}$ be the second face. Let w be a vertex of C_f that belongs to $\bar{f}$. Since {u,v} is not an edge, w exists. Hence {u,v} is a separation pair. ●

We are ready to state the overall strategy.

Algorithm Separation Pairs

1) Find a separating cycle C.

2) Find separation pairs on C (*Cycle Separation Pairs*).

3) Find separation pairs {u,v} such that u is on C and v is on a bridge relative to C (*Cycle-Bridge Separation Pairs*).

4) Apply the algorithm recursively to each bridge with its vertices of attachment collapsed to a big vertex. Remove all the separation pairs that include a big vertex.

Before elaborating on how to do steps 2 and 3, we state the following lemma that establishes the correctness of the above algorithm given that we can do steps 2 and 3 correctly.

Lemma2: Let C be a cycle in a planar biconnected graph G, and let B be a bridge of G relative to C with vertices of attachment $u_1, u_2, \ldots, u_k$, $k \geq 2$. Let $\bar{B}$ be obtained from B by collapsing $u_1, u_2, \ldots, u_k$ into a single vertex S. If {x,y} is a separation pair of G, x and y are in B but neither is in C, then

{x,y} is a separation pair in $\bar{B}$.

Proof: Since {x,y} is a separation pair, x and y must belong to at least 2 or 3 distinct faces. Notice that if {x,y} is not an edge in B, {x,y} will not be an edge in $\bar{B}$. Hence the lemma follows. ●

We now show how to do each of steps 2 and 3. Before we need to introduce some more terms.

Let $B_1,B_2,...,B_t$ be the bridges relative to a cycle C of G. The *segments* of B_i are the partitions of C induced by the vertices of attachments of B_i. Two bridges *conflict* if the vertices of attachment of one are contained in at least two segments of the other. The transitive closure of the relation "conflict" yields the *big bridges* of C. We are now ready for describing the implementation of step 2 above.

Algorithm Cycle Separation Pairs

Input: A planar embedding of a planar biconnected graph G and a separating cycle C.

Output: A minimal complete set of separation pairs which lie on the cycle C.

1) Find all the bridges of G relative to C.

2) Collect all adjacent degree two vertices on C and identify a corresponding set of separation pairs. Remove all such vertices and replace each adjacent set by a virtual edge.
Remark: Let $x_0,x_1,...,x_t,x_{t+1}$ be a maximal set of adjacent vertices (in order) on C such that x_i is of degree two, $1 \leq i \leq t$. Then output $\{x_0,x_i\}$, $i \geq 2$ as separation pairs, remove all the degree two vertices, and add the virtual edge $\{x_0,x_{t+1}\}$.

3) For each bridge that has exactly two vertices of attachment u and v and

Claim: The segments induced by u and v are not connected by a path outside C if and only if the vertices of attachment of any big bridge belong only to one of the segments.

It follows from the above that if u and v to the same bridge they will be identified as a separation pair in step 6. Otherwise step 9 will either determine that {u,v} is a separation pair or that u and v belong to two distinct bridges relative to some other identified separation pair.

Finding the bridges and the big bridges can be done in $O(\sqrt{n})$ time following the connected components strategy .If we make a rooted tree out of C (with an extra edge from the root to a node), we can implement steps 6-9 efficiently. ●

We now show how to do step 3 of the algorithm **Separation Pairs.** The following fact will be needed.

Lemma 4: Let C be a cycle and let B be an outerbridge of C with vertices of attachment $u_1, u_2, \ldots, u_k$, $k \geq 2$. Suppose that $\{u_i, u\}$ and $\{u_j, v\}$ $(i \neq j)$ are separation pairs such that u_j and v are in two distinct bridges of the edges in B and C relative to $\{u_i, u\}$. Then there exists a face f containing u_i, u_j, u and v.

Proof: Suppose that $\{u_i, u\}$ is not an edge. Consider the interior faces $f_1, f_2, \ldots, f_t$ containing u_i and u. Each bridge relative to $\{u_i, u\}$ must consist of a segment q determined by $\{u_i, u\}$ in one of the faces f_i's plus all the faces sharing an edge with q and all faces connected to these faces in the dual graph. u_j and v must be in one face. Hence u_j and u must be in one of the

f_i' s. ●

Algorithm Cycle-Bridge Separation Pairs

Input: A planar embedding of a planar graph G and a separating cycle C.

Output: The set of separation pairs with one endpoint on C and the other outside C.

1) Identify the set F of interior faces that contain a vertex on C and another outside C.

2) For each face f in F, create the triplets (u,v,f), where u is a vertex of attachment in f, v is a vertex of f but not on C, and u and v belong to a bridge. Notice that any face f in F can contain at most two vertices of attachment in a single bridge.

3) Sort the triplets {(u,v,f)}.

4) Identify pairs {u,v} such that there are two or three triplets of the form (u,v,f_i), $2 \leq i \leq 3$. Identify the corresponding separation pairs.

5) For each face f in F with two vertices of attachment u_1 and u_2, and containing separation pairs $\{u_1, v_i\}$ $(1 \leq i \leq s)$ and $\{u_2, w_k\}$, $(1 \leq k \leq t)$, reduce the set of separation pairs to a minimal complete set (i.e. no pair will be contained in two distinct bridges with respect to any other pair).

Lemma5: The above algorithm correctly finds the separation pairs {u,v} such that u is on C and v is on a bridge relative to C. The algorithm runs in time $O(\sqrt{n})$.

Proof: By theorem 1 a separation pair must belong to two or three distinct faces. Steps 1-4 of the above algorithm determine all such pairs such that one vertex is on C and the other is on a bridge. Based on Lemma4, step 5 correctly eliminates the redundant separation pairs. It is a simple matter to

establish the running time. ●

Theorem3: Given a planar embedding of a planar graph, it is possible to determine a complete set of separation pairs in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh of processors.

The main problem with determining the triconnected components using a separating cycle is that we need to identify the triconnected pieces of each bridge *plus* the segments of C determined by the bridge. However, the resulting straightforward divide-and-conquer approach algorithm will be inefficient. For the separating cycle found by our algorithm, we need only to worry about outer bridges. What we do is to collapse the vertices of attachment of each outer bridge into a *big vertex*. In this case, the triconnected components of an outer bridge may have been altered. We handle this problem as follows.

Algorithm Split Components of a Modified Bridge

Input: A bridge B of a planar graph G relative to a cycle C with its segments on C and the split components of B when all its vertices of attachment have been collapsed into a big vertex S.

Output: The split components of the subgraph of G determined by B and its corresponding segments on C.

1) If B has exactly two vertices of attachment with C, then replace C with a virtual edge connecting the two vertices of attachment.

2) Remove all the split components that do not contain the big vertex S. Add a virtual edge for every separation pair in the remaining piece. Let B be the resulting subgraph.

3) Remove all edges e=(v,w) such that {v,w} is a separation pair. Identify the

correct new faces of $\bar{B}$.

4) For each face f of $\bar{B}$ containing at least one separation pair, do

If f contains one vertex of attachment v and the separation pairs $\{v,u_1\}$, $\{v,u_2\}$,..., $\{v,u_k\}$ (See figure below), then create k+1 *new faces* as shown in the figure. Each f_i consists of virtual edges (v,u_{i-1}) and (v,u_i) plus segment $[u_{i-1},u_i]$ of the cycle C_f determined by f. (v,u_{i-1}) and (v,u_i) belong to f_i and to the exterior face, $1<i<k+1$. Similarly we can define f_1 and f_{k+1} each containing one virtual edge only.

If f contains two vertices of attachment, then a similar scheme will work.

5) Remove duplicate pieces (i.e. pieces that have the same sets of edges).

6) Find the connected pieces of the dual of the subgrpahs obtained. Each such a piece determines a splitting component. Put back all the edges whose endpoints form a separation pair.

Lemma6: The above algorithm correctly identifies the splitting components of the subgraph determined by the bridge B and its segments on the cycle C.

Proof: It is clear from Lemma2 that the modifications done in step 1 will not change the splitting components of the subgraph. Now observe the following facts:

Fact1: There exist no separation pairs with neither endpoint on C.

Fact2: Let {u,v} be a separation pair such that $u \in C$. Then for any splitting corresponding to {u,v}, one split component will contain no vertex of C except u.

It is clear that the splitting process performed at step 4 is legal with the exception of creating duplicate split components. Step 5 takes care of the duplicates. Step 6 identifies the pieces obtained after the splitting process.

●

It is easy to deduce the triconneted components once the splitting components have been identified. The overall algorithm is given below.

Algorithm Triconnected Components

Input: An embedding of a planar biconnected graph G=(V,E).

Output: The triconnected components of G.

1) If the graph is a set of multiple edges, or a triangle or empty, then exit. Else do the following.

2) Find a partitioning cycle C and identify the corresponding cycle and cycle-bridge separation pairs.

3) Identify the triconnected pieces induced in steps 2,4, and 9 of algorithm **Cycle Separation Pairs.**

4) Split off each big bridge and add to it the appropriate virtual edges.

5) Decompose each subgraph introduced in step 3 into inner and outer parts, putting the corresponding segments of C in the inner parts. For each outer part that does not consist only of multiple edges, collapse the vertices of attachment into one big vertex.

6) For each of the subgraphs formed in step 4, recursively find their triconnected components

7) Apply algorithm Split Components of a Modified Bridge to obtain the correct triconnected components of each outerbridge.

8)All the triconnected components with exactly two vertices of attachment should be removed (and identified as triconnected pieces) and a virtual edge should connect the two vertices of attachment.

9) Identify the triconnected components that conflict with each other. Find the transitive closure of the relation "conflict" using the connected components strategy.

10) Merge multiple edges and triangles as much as possible. The resulting pieces from this and previous steps are the triconnected components of G.

Theorem4: The above algorithm correctly identifies the triconnected components of a planar biconnected graph in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ array of processors. On a PRAM, this algorithm takes $O(log^3 n)$ time with $O(n)$ processors.

5. Isomorphism of Triconnected Planar Graphs

Whitney ([14]) has shown that a triconnected planar graph has two embeddings in the plane such that for each vertex v the order of the edges around v in one embedding is the reverse of the order of the edges around v in the other embedding. Therefore two triconnected planar graphs G_1 and G_2 are isomorphic if a plane embedding of G_1 is isomorphic to one of the two planar embeddings of G_2. Let's direct the edges of G_1 and G_2 by replacing each undirected edge by the corresponding two directed arcs. For the rest of this section, we will assume that G_1 and G_2 represent the corresponding directed graphs.

Let e_1 and e_2 be two arcs of G_1 such that $e_1 = (a,b)$ and $e_2 = (b,c)$. We define $d(e_1,e_2) = i$ if e_2 is the ith arc in a clockwise ordering of the arcs out of b such that the first arc is the one to the immediate left of e_1. Let $P=\{e_1,e_2,...,e_t\}$ be a directed path in G_1 such that no two arcs are the same. A path $P'=\{f_1,f_2,...,f_{t'}\}$ is *similar* to P if t=t' and $d(e_i,e_{i+1}) = d(f_i,f_{i+1})$, $i=1,2,...,t-1$. The isomorphism algorithm consists of (i) finding an arbitrary Euler circuit C in G_1, (ii) identifying a set of similar circuits $\{Q_i\}$ in G_2, and (iii) checking whether C and one of the Q_i's induce an isomorphism between G_1 and G_2. We provide the details below.

Algorithm isomorphism of triconnected planar graphs

Input: Two triconnected planar graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with their embeddings.

Output: An isomorphism between G_1 and G_2 if the two graphs are isomorphic.

1. Find an arbitrary Euler circuit C in G_1. Starting with an arbitrary arc $e = e_1$, let $e_1, e_2, \ldots, e_t$ be the sequence of arcs of G_1 in the order given by C. Determine the sequence of positive intergers $i_1, i_2, \ldots, i_t$ such that $i_j = d(e_j, e_{j+1})$, $1 \leq j \leq t-1$ and $i_t = d(e_t, e_1)$.

2. Identify similar Euler circuits in G_2 as follows. Assume without loss of generality that t is a power of 2. If $t \leq 16$, then use any brute force method. Else, find the paths in G_2 with the corresponding sequences $i_1, i_2, \ldots, i_{\frac{t}{2}-1}$ and $i_{\frac{t}{2}+1}, \ldots, i_{t-1}$. Two such paths P_1 and P_2 can be combined if (1) no two arcs in P_1 and P_2 are the same, (2) the first arc on P_2 is the $i_{\frac{t}{2}}$th arc following the last arc on P_1, and (3) the i_t th arc of the last arc on P_2 is the first arc on P_1. If no two such paths could be combined then stop the two graphs are not isomorphic.

3. Let $Q_1, Q_2, \ldots, Q_s$ be the paths obtained from step 2. Number the vertices in the order they occur in C and the Q_is. Two identical sequences define an isomorphism.

Theorem5 : The above algorithm tests whether two triconnected planar graphs are isomorphic on a PRAM with $O(n^2)$ processors in time $O(log^2 n)$.

Proof: Using the algorithms in ([3],[4]) the Euler circuit C can be found in $O(log^2 n)$ time using $O(n)$ processors. The sequence $\{i_j\}$ can be determined within these bounds from the planar representation. The recursion of step 2 runs $O(logn)$ times such that the merging in each iteration takes $O(logn)$ time and requires $O(n^2)$ processors. Notice that the number of paths arising in each iteration is $O(n)$ since similar paths with the same first arc are identical. It is not hard to see that step 3 can be done in $O(log^2 n)$ time with $O(n)$ processors. ●

6. The Single Function Coarsest Partitioning Problem

Given a set S={1,2,...,n}, an initial partition of S, $B=\{B_1,...,B_m\}$, and a function $f: S \rightarrow S$, we want to find the equivalence classes of the following relation Δ :

$a \Delta B$ *if* $f^i(a)$ and $f^i(b)$ *are in the same partition* for *all* i

We can translate this to the following graph problem. Create n nodes, named 1,2,...,n, and label each vertex i by j if i belongs to B_j. For every i, create the directed edge (i,f(i)). In this node labelled graph G, the outdegree of each node is 1. Any two nodes a and b are equivalent if and only if for every i, the length i paths from a and b terminate at nodes with the same label. We

want to relabel the nodes so that any two nodes get the same label if and only if they are equivalent.

Now we briefly outline the main steps of the algorithm. Note that G consists of one or more components (undirected version) and each component consists of a tree and an additional edge.

Algorithm Coarsest Partitioning

Input: S=$\{1,2,...,n\}$, a partition of S, $\{B_1,B_2, \ldots, B_m\}$ and the function f, $\{(i,f(i)) \mid 1 \leq i \leq n\}$.

Output: label the elements of S such that two elements get the same label if and only if they are equivalent.

1) Find the spanning forest of G (undirected version) [12]. Note that one edge from each component will be excluded.

2) If the excluded edges of the directed G are $(r_1,u_1),...,(r_m,u_m)$ then choose the r_i's as the roots of the trees.

3) Mark the nodes on each tree path from u_i to r_i.

4) We have isolated m cycles, one for each G_i. Let the ith cycle be $u_{i_1}(=u_i)u_{i_2} \cdots u_{i_j}(=r_i)$. Move a copy of each cycle into a separate area. In the ith area find the smallest t such that there exists a $k \geq 1$ satisfying $(u_{i_1}u_{i_2} \cdots u_{i_t})^k = u_{i_1}u_{i_2} \cdots u_{i_j}$. Denote $u_{i_1}...u_{i_t}$ as period. Fold so that the

new cycle of G_i is $u_{i_1}u_{i_2}...u_{i_t}$. At this stage we can assume that for every G_i its cycle and the period of its cycle are the same.

5) Define two cycles as *cyclic shift equivalent* iff one is a cyclic shift of the other. Partition the cycles of the G_i's into equivalence classes.

6) "Merge" equivalent cycles (and the corresponding G_i's become connected). Relabel the nodes so that any two nodes in two different components get different labels; however any two nodes in the same component will have the same label if they had the same labels before.

7) Now each connected component can be considered separately.

It is not hard to see how to implement the above steps except steps 4 and 5 in $O(\sqrt{n})$ time. Step 5 can be done within the same time bound by using the pattern matching algorithm of [5] to test whether two cycles are cyclic shift equivalent. Below we show how to compute the period of step 4 quickly. The algorithm presented was discovered jointly with S. Krishnamurthy and is a simpler version of the authors' original algorithm. Before we present the algorithm we need a couple of lemmas.

Let x be a given string over a certain alphabet. y is *period* of x if x is a concatenation of k copies of y for some positive integer k. *The* period of x is the shortest such y.

Lemma7: Let a string x be split into two parts y and z. If $yz = zy$, then

both y and z have a common period.

Proof: Without loss of generality assume that $|z| > |y|$. Clearly z consists of y followed by some string w. Substituting yw for z in $yz = zy$, we obtain that y and w commute. We can continue in this fashion until we obtain two substrings that are identical. One can show by induction that the final substring is a period of both y and z. ●

Lemma8: Let x be a string of length n and let n_1 and n_2 be divisors of n such that $n_1 > n_2$ and n_2 does not divide n_1. Suppose that $x = yyw$, where $|y| = n_1$ and that y is not a period of x. Then the prefix of x of length n_2 cannot be a period of x.

Proof: Let z be the prefix of x of length n_2. Suppose that z is a period of x. Then one can check that z can be written as $z = pq = qp$, for some substrings p and q. Using Lemma7 we find that y and z have the same period which is a contradiction to the fact that y is not a period of x. ●

We are ready to state our algorithm. If i is a given positive integer, we use x_i to denote the prefix of x of length i.

Algorithm Finding the Period

Input: A string x and its length n.

Output: The period y of x.

1. Find all divisors $d_1 < d_2 < \ldots < d_k$ of n.

2. Check if x_{d_k} is a period. If yes, repeat the procedure recursively on this substring. Otherwise go to step 3.

3. Go down the divisors until the largest i is found such that $x_{d_i} x_{d_i}$ is a prefix of x. If no such i exists, then x is the period. Else, check if x_{d_i} is a period. If yes, repeat the procedure recursively on this substring. Otherwise, output x as the period.

Theorem6: The above algorithm correctly finds the period of the string x in time $O(\sqrt{n})$ on a $\sqrt{n} \times \sqrt{n}$ mesh.

Proof: Let y be the period of x and let $|y| = d_t$. Let d_p be the largest multiple of d_t among the divisors of n. Using Lemma8 it is clear that x_{d_p} will be the first substring to satisfy the condition in step 3 of the algorithm. The recursive application of the algorithm will go down to y and stop there.

The running time of the algorithm can be easily shown by observing that the input length at each recursive call is a constant fraction of the previous input length and that the time it takes to execute each step is optimal in the input length. ●

One can easily develop an NC algorithm for the multiple function coarsest partitioning problem by creating equivalence sets for each pair of elements and applying standard merging procedure for log n iterations. However

the number of processors involved is $O(n^4)$. This yields an $O(log^2 n)$ algorithm with $O(n^4)$ processors for minimizing the number of states of a finite automaton or determining whether two automata are equivalent[1]. An interesting open problem is whether there exist fast parallel algorithms for the multiple function coarsest partitioning problem using only $O(n)$ processors.

[1]Better processor bounds can be obtained for these two problems.

References

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

2. M. Atallah and S. Hambrusch, "Solving tree problems on a mesh-connected processor array", Proceedings of 26th IEEE Symp. FOCS, 1985, pp. 222-231.

3. B. Awerbuch, A. Israeli and Y. Shiloach," Finding Euler circuits in logarithmic parallel time," Proceedings of the 16th Annual Symposuim on Theory of Computing, pp. 249-257, Washington, D.C., 1984.

4. M. Atallah and U. Vishkin, "Finding Euler tours in parallel," JCSS 29,3(1984),330-337.

5. Z. Galil, "Optimal Parallel algorithms for string matchings," Proceedings of the 1984 STOC, pp. 240-248.

6. J. Hopcroft and R. Tarjan," Dividing a graph into triconnected components," SIAM J. on Computing, 1973, pp. 135-158.

7. J. Hopcroft and R. Tarjan," A VlogV algorithm for isomorphism of triconnected planar graphs," JCSS, 1973, pp. 323-333.

8. J. Hopcroft and R. Tarjan," A V^2 algorithm for determining isomorphism of planar graphs," IPL, 1971, pp. 32-34.

9. J. Ja'Ja' and J. Simon," Parallel Algorithms in Graph theory: planarity testing," SIAM J. Computing, 1982, pp. 314-328.

10. G. Miller and V. Ramachandran, "A New graph triconnectivity algorithm

and its parallelization," to appear.

11. G. Miller and J. Reif," Parallel tree contraction and its application," Proceedings of 26th IEEE Symp. FOCS,1985, pp. 478-489.

12. Q. Stout, "Tree-based Graph Algorithms for Some Parallel Computers," Proceedings 1985 Int. Conf. on Parallel Computing, 1985, pp. 727-733

13. J. Smith, "Parallel Algorithms for Depth-First Searches I. Planar Graphs," SICOMP, 15(3), pp. 814-830, August 1986.

14. H. Witney, "A set of topological invariants for graphs," American Journal Math 55, pp. 321-335, 1937.