Chapter in SYNTHESIS OF PARALLEL ALGORITHMS, Morgan-Kaufmann, 1993. (Figures not yet included in this file.)

Parallel Open Ear Decomposition with Applications to Graph Biconnectivity and Triconnectivity *

Vijaya Ramachandran
Department of Computer Sciences
University of Texas
Austin, TX 78712

January 20, 1992

Abstract

This report deals with a parallel algorithmic technique that has proved to be very useful in the design of efficient parallel algorithms for several problems on undirected graphs. We describe this method for searching undirected graphs, called "open ear decomposition", and we relate this decomposition to graph biconnectivity. We present an efficient parallel algorithm for finding this decomposition and we relate it to a sequential algorithm based on depth-first search. We then apply open ear decomposition to obtain an efficient parallel algorithm for testing graph triconnectivity and for finding the triconnnected components of a graph.

This material will appear as a chapter in the book, *Synthesis of Parallel Algorithms*, edited by John Reif, which is to be published by Morgan-Kaufmann.

1 Introduction

In this report we introduce *open ear decomposition,* which is a method for searching an undirected graph. We present an algorithm that either finds an open ear decomposition in an undirected graph or reports that no open ear decomposition exists. This algorithm runs in logarithmic time with a linear number of processors. A graph has an open ear decomposition if and only if it is biconnected. Hence this algorithm allows us to determine graph biconnectivity efficiently in logarithmic parallel time.

*This work was supported in part by NSF Grant CCR-8910707

We use open ear decomposition to obtain a logarithmic time parallel algorithm using a linear number of processors to find the triconnected components of a graph. This algorithm is fairly complex and we present it in a top-down manner by first giving the high-level ideas leading to the algorithm and then giving efficient implementations of the various steps. In the last section we give some pointers towards obtaining optimal logarithmic time parallel algorithms for graph biconnectivity and triconnectivity.

Open ear decomposition has been used to obtain efficient parallel algorithms for several other important graph problems such as graph four-connectivity [KR91], *st*-numbering [MSV86] and graph planarity [RR89].

Algorithmic Notation

The algorithmic notation in this report is from Tarjan [Ta83]. We enclose comments between a pair of curly brackets with asterisks ('{*' and '*}'). We incorporate parallelism by use of the following statement that augments the **for** statement.

pfor iterator $\rightarrow$ statement list **rofp**

The effect of this statement is to perform the **pfor** loop in parallel for each value of the iterator.

2 Ear Decomposition and Two-Connectivity

In this section we define *ear decomposition* and *open ear decomposition* and relate these to graph *two-edge-connectivity* and *two-vertex-connectivity* (i.e., *biconnectivity*). We then describe efficient parallel algorithms to find these decompositions. We also relate these parallel algorithms to the classical sequential algorithm for testing graph biconnectivity, which is based on depth-first search.

2.1 Basic Definitions

An *undirected graph* G is a pair $(V,\ E)$ where V is the set of *vertices* of G and E is the set of *edges* of G; an edge is an unordered pair of distinct vertices. We denote the undirected graph by $G = (V, E)$ and we sometimes refer to it as G. An edge (u, v) is *incident* on vertices u and v. Vertices u and v are *adjacent* in G if G contains edge (u, v). The *degree* of a vertex is the number of edges incident on the vertex. We will sometimes refer to an undirected graph as simply a *graph.*

A *directed graph* $G = (V, E)$ consists of a vertex set V and an edge set E containing ordered pairs of elements from V. An edge (u, v) in a directed graph is directed from u to v and is *outgoing* from u and *incoming* to v.

A *multigraph* G is a pair (V, E) where V is the set of vertices of G and E is the *multiset* of edges of G; an edge of a multigraph is an unordered pair of vertices. We allow edges of the form $(v, v), v \in V$ and we call such edges *self-loops.* An edge e in a multigraph may be denoted by (a, b, i) to distinguish it from other edges between a

and b; in such cases the third entry in the triplet may be omitted for one of the edges between a and b.

A *path* P in G is a sequence of vertices $\langle v_0, ..., v_k\rangle$ such that $(v_{i-1}, v_i) \in E, i = 1, ..., k$; P is directed or undirected depending on whether G is directed or undirected. The path P *contains* the vertices $v_0, ..., v_k$ and the edges $(v_0, v_1), ..., (v_{k-1}, v_k)$ and has *endpoints* v_0, v_k, and *internal vertices* $v_1, ..., v_{k-1}$. The path P is a *simple path* if $v_0, ..., v_{k-1}$ are distinct and $v_1, ..., v_k$ are distinct, and all edges on P are distinct. A simple path $P = \langle v_0, ..., v_k\rangle$ is a *simple cycle* if $v_0 = v_k$; otherwise P is *noncyclic.* The path $\langle v\rangle$ is a *trivial path* with no edges.

A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The *subgraph of G induced by V'* is the graph $H = (V', F)$ where $F = \{(u, v) \in E \mid u, v \in V'\}$.

An undirected graph $G = (V, E)$ is *connected* if there exists a path between every pair of vertices in V. A *connected component* of a graph G is a maximal induced subgraph of G which is connected.

Let $G = (V, E)$ and $H = (W, F)$ be a pair of graphs. The graph $G \cup H$ is the graph $G' = (V \cup W, E \cup F)$. If $W \subseteq V$ then the graph $G - H$ is the graph $H' = (V, E - F)$.

A *tree* is a connected graph containing no cycle. A *leaf* in a tree is a vertex of degree 1. Let $T = (V, E)$ be a tree and let $r \in V$. The *out-tree* $T = (V, E, r)$ *rooted at* r (or simply the *tree T rooted at r)* is the directed graph obtained from T by directing each edge such that every path from r to any other vertex is directed away from r. The *in-tree rooted at* r is the directed graph obtained from T by directing each edge such that the path from every vertex to r is directed towards r.

Let (x, y) be a directed edge in a rooted tree T. Then, x is the *parent* of y and y is a *child* of x in T. Vertex v is a *descendant* of vertex u (and equivalently, u is an *ancestor* of v) if there is a directed path from u to v in T. Vertex v is a *proper descendant* of u (and u a *proper ancestor* of v) if v is a descendant of u and $u \neq v$. Given a pair of vertices $u, v \in V$, the *least common ancestor of u and v,* denoted by $lca(u, v)$ is the vertex $w \in V$ that is an ancestor of both u and v with no child of w being an ancestor of both u and v. For an edge $e = (u, v)$ the *least common ancestor of e,* denoted by $lca(e)$, is the vertex $lca(u, v)$.

A *preorder* labeling of the vertices of a rooted tree T labels the root of T and then the vertices in the subtree rooted at each child of the root in turn.

Let $G = (V, E)$ be a connected graph. A spanning tree T of G is a subgraph of G with vertex set V such that T is a tree. An edge in $G - T$ is a *nontree edge with respect to T.*

Let T be a spanning tree of G. Any nontree edge e of G creates a cycle in the graph $T \cup \{e\}$, called the *fundamental cycle of e with respect to T.* Let $r \in V$, and let T be rooted at r.

Let $e = (u, v)$ be a nontree edge in $T = (V, E, r)$ and let $lca(e) = l$. The fundamental cycle of e with respect to T consists of the path from l to u, followed by edge e, followed by the path from v to l. Let (l, a) be the first edge on the path from l to u

and (l, b) be the first edge on the path from l to v (it is possible for one of these edges to be missing). Then edges (l, a) and (l, b) are the *base edge(s) of the fundamental cycle of* e (when they exist) and the vertices a and b are the *base vertice(s) of the fundamental cycle of* e (when they exist).

An edge $e \in E$ in a connected graph $G = (V, E)$ is a *cutedge* if e does not lie on a cycle in G. A connected undirected graph $G = (V, E)$ is *2-edge connected* if it contains no cutedge. A *2-edge connected component of* G is a maximal induced subgraph of G which is 2-edge connected.

A vertex $v \in V$ is a *cutpoint* of a connected undirected graph $G = (V, E)$ if the subgraph induced by $V - \{v\}$ is not connected. A connected graph G is *biconnected* (or *two-vertex connected)* if it contains at least 3 vertices and has no cutpoint. A *biconnected component* (or *block*) of G is a maximal induced subgraph of G which is biconnected.

By *Menger's theorem* a graph is 2-edge connected if and only if there are at least two edge-disjoint paths between every pair of distinct vertices, and a graph is biconnected if and only if the graph is connected and has no more than two vertices or there are at least two vertex-disjoint paths between every pair of distinct vertices.

The *two-connectivity* problem is the problem of determining 2-edge connectivity and biconnectivity in a connected graph.

2.2 Ear Decomposition

An *ear decomposition* $D = [P_0, P_1, ..., P_{r-1}]$ of an undirected graph $G = (V, E)$ is a partition of E into an ordered collection of edge-disjoint simple paths $P_0, ..., P_{r-1}$ such that P_0 is an edge, $P_0 \cup P_1$ is a simple cycle, and each endpoint of P_i, for $i > 1$, is contained in some $P_j, j < i$, and none of the internal vertices of P_i are contained in any $P_j, j < i$. The paths in D are called *ears.* An ear is *open* if it is noncyclic and is *closed* otherwise. A *trivial ear* is an ear containing a single edge. D is an *open ear decomposition* if all of its ears are open.

Let $D = [P_0, ..., P_{r-1}]$ be an ear decomposition for a graph $G = (V, E)$. For a vertex v in V, we denote by $ear(v)$, the index of the lowest-numbered ear that contains v; for an edge $e = (x, y)$ in E, we denote by $ear(e)$ (or $ear(x, y)$), the index of the unique ear that contains e. A vertex v *belongs to* $P_{ear(v)}$.

Lemma 2.1 [Wh32] An undirected graph $G = (V, E)$ has an ear decomposition if and only if G is 2-edge connected.

Proof We first prove the *if* part of the lemma. Assume G is 2-edge connected. We construct an ear decomposition for G as follows. To construct P_0 and P_1, we pick any edge $e = (u, v)$ in G. Since e is not a cutedge, there is a simple path between u and v in G that avoids e. Let P be such a path. We construct P_0 as $\langle e \rangle$ and P_1 as P. Then P_0 is an edge and $P_0 \cup P_1$ is a simple cycle as required.

Assume inductively that we have constructed $H_{i-1} = \cup_{j=0}^{i-1} P_j, i > 1$. To construct P_i, we pick an edge (x, y) that is not contained in H_{i-1} but with vertex x in H_{i-1}.

We then find a simple path Q from y to x in G that avoids edge (x, y). Let z be the first vertex on path Q that is contained in H_{i-1}. We construct P_i as the edge (x, y) followed by the path Q from y to z. This path has each of its endpoints on some $P_j, j < i$, and none of its internal vertices on any $P_j, j < i$. Hence it is an ear.

We now prove the *only if* part. Let $D = [P_0, ..., P_{r-1}]$ be an ear decomposition for G. We will prove by induction on i for $i > 0$ that the graph $H_i = \cup_{j=0}^{i} P_j$ is 2-edge connected. For the base case, $P_0 \cup P_1$ is a simple cycle, and therefore H_1 is 2-edge connected.

Assume inductively that H_{i-1} is 2-edge connected and consider H_i. To show that H_i is 2-edge connected it suffices to show that every edge on P_i lies on a cycle. Let the endpoints of P_i be x and y and let Q be a path from x to y in H_{i-1}. The path Q exists since H_{i-1} is connected. Every edge on P_i lies on the cycle $P_i \cup Q$ in H_i and hence H_i is 2-edge connected.[]

Lemma 2.2 [Wh32] A graph has an open ear decomposition if and only if it is biconnected.
Proof Exercise 1.[]

2.3 An Efficient Parallel Algorithm for Ear Decomposition

In this section we present an efficient parallel algorithm for finding an ear decomposition for a 2-edge connected graph. This algorithm is from [MR86] and [MSV86], and is an efficient parallel implementation of an algorithm in [Lo85].

Algorithm 2.1: Ear Decomposition Algorithm

Input: A 2-edge connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$.
Output A numbering on the edges in E, specifying their ear number.

vertex v, r; **edge** e;

1. {$*$ Preprocess. $*$} find a spanning tree T for G, pick a root vertex r and number the vertices of T in preorder from 0 to $n-1$ with respect to root r;

2. {$*$ Assign ear numbers to nontree edges in T. $*$}

   2a. label each nontree edge e in G by its least common ancestor $lca(e)$ in T;

   2b. sort the labels of nontree edges in nondecreasing order and relabel them in order as 1, 2, ...;

3. {$*$ Extend the numbering assigned in step 2 to the tree edges by numbering each tree edge t by the label of the nontree edge with smallest label whose fundamental cycle contains t. $*$}

3a. label each vertex with the label of the nontree edge incident on it with the minimum label;

3b. assign to each tree edge $(parent(v), v)$ in T, the label of the minimum label of any descendent of v (including v);

4. relabel the nontree edge labeled 1 by the label 0

end.

We now prove the correctness of Algorithm 2.1 and then provide implementation details.

Lemma 2.3 Algorithm 2.1 obtains an ear decomposition of a 2-edge connected graph.

Proof We first observe that the label given to tree edge $t = (parent(v),\ v)$ in step 3b is the label of the nontree edge with smallest label whose fundamental cycle contains t. This is because any such nontree edge e must be incident on a descendant of v, and any nontree edge n incident on a descendant of v with $lca(n)\ \leq\ v$ must include edge t in its fundamental cycle.

We now prove by induction on i that the edges with label i form a simple path that satisfies the definition of ear P_i.

BASE: P_0 and P_1. Let e be the nontree edge given label 1 in step 2b. Then by step 3 every tree edge in the fundamental cycle of e will be assigned label 1. Further any tree edge not on the fundamental cycle of e will be assigned a label greater than 1. Hence the edges labeled 1 at the end of step 3 are exactly the edges in the fundamental cycle of e and these form a simple cycle as required for $P_0 \cup P_1$. By step 4 the label of e is set to be 0. Hence $P_0 = \{e\}$ and P_1 becomes a simple noncyclic path with its two endpoints on e.

INDUCTION STEP: Assume the result is true for up to P_{i-1}, $i > 1$, and consider the nontree edge $f = (u, v)$ with label i. Let $lca(f) = l$. Hence the tree edges in the fundamental cycle of f are the edges on the tree path P from l to u and on the tree path Q from l to v.

Consider the tree path P. Assume that P contains at least one edge with label $j \neq i$ and let (x, y) be the first edge on $R = P \cup \{f\}$ that has label i. We claim that every edge on R from x to v has label i and every edge in P from l to x has label less than i. To see the first part of the claim we note that by step 3 f is the nontree edge with smallest label whose fundamental cycle contains tree edge (x, y). Every edge on P from y to u lies on the fundamental cycle of f, so if any edge on this path does not have label i then it must have a label $j < i$. But then the nontree edge g with label j has $lca(g) \leq l$ by the labeling in step 2b. But then, edge (x, y) would be in the fundamental cycle of g and would be labeled j rather than i, which is a contradiction. Hence every edge on P from x to u is labeled i. Finally, edge (u, v) is labeled i by assumption. Hence all edges on R from x to v have label i.

To see the second part of the claim, consider tree edge $s = (x, parent(x))$. Since by assumption the edge s has a label j that is different from i, we know that tree edge s lies on the fundamental cycle of a nontree edge h with label j and that $j < i$. Further since $j < i$ we must have $lca(h) \leq l$ and hence every edge on the path P from l to x lies on the fundamental cycle of h. Hence the label of every edge on P from l to x is at most j and hence is less than i.

A similar argument holds for the path Q for the case when Q contains at least one edge with label $j \neq i$. Hence the edges with label i form a simple path that consists of a portion of tree path P starting at some vertex x and extending up to u, followed by edge (u, v) followed by a portion of the tree path Q from v to some vertex $z > l$; further the two endpoints of this path are contained in ears numbered lower than i.

Finally, if P or Q contains no edge with label $j \neq i$ then we note that the label of tree edge $(parent(l), l)$ is less than i since any nontree edge g whose fundamental cycle contains this tree edge has $lca(g) < l$. Further, such a nontree edge g must exist since the graph is 2-edge connected. Hence vertex l is contained in an ear P_k with $k < i$ and hence the endpoints of ear P_i are contained on an ear with label smaller than i.[]

Let us analyze the complexity of Algorithm 2.1.
Step 1 requires the computation of a spanning tree T and its preorder numbering with respect to the root r [CV86].
Step 2a requires the computation of least common ancestors in T [SV88].
Step 2b requires sorting of integers in the range $[0..n-1]$ [C88].
Step 3a requires the computation of the minimum value in each adjacency list [KR90].
Step 3b can be performed efficiently in parallel by the following simple method using the Euler tour technique on trees [TV84]. Note that the vertices that are the descendants of a vertex v in the tree T lie between the first and last occurrences of v in the Euler tour of T. In step 3b we need to compute the minimum value in each such interval. For this we first build a table of such minimum values for all intervals of length $2^i, 0 \leq i \leq logn$. This table can be constructed in $O(logn)$ time using n processors. Once we have this table, the minimum value for any other interval I can be computed from the precomputed minimum values of two overlapping intervals whose union gives I. This part of the computation can be performed in constant time using one processor for each interval.
Step 4 is trivial to implement.

As seen above all of the steps in Algorithm 2.1 can be performed in logarithmic time with a linear number of processors using well-known efficient parallel algorithms. We also leave it as an exercise for the reader to verify that Algorithm 2.1 runs in linear sequential time.

2.4 Ear Decomposition and Depth-First Search

Algorithm 2.1 of the previous section computes an ear decomposition of a graph in linear sequential time. The computation in Algorithm 2.1 can be simplified considerably in the sequential algorithm if the spanning tree T is a depth-first search tree rooted at r. In that case, the lca computation in step 2a is immediate, since every nontree edge in the depth-first search tree goes from a vertex to its ancestor, and this ancestor will be the lca. We defer step 2b to the end of the algorithm and to compute step 3, we define the following two functions on vertices. (We assume that the vertices are numbered in preorder, starting with 0, and that the input graph has n nodes.)

$low(v) = \min(\{w | w$ lies on the fundamental cycle of a nontree edge incident on a descendant of $v\} \cup \{n\})$

$ear(v) = lexmin(\{(w,x) | (w,x)$ is a nontree edge with x a descendant of $v\} \cup \{(n,n)\})$

The values $low(v)$ and $ear(v)$ can be computed incrementally during the depth-first search of G that generates T. This is given in Algorithm 2.2 below. Note that Algorithm 2.2 is essentially the well-known linear time sequential algorithm for graph biconnectivity [Ta72].

Algorithm 2.2: Sequential Ear Decomposition Algorithm
Input: A connected graph $G = (V, E)$ with a root $r \in V$, and with $|V| = n$.
Output: A depth-first search tree of G, together with a label on each edge in E, indicating its ear number.

set T **of edges** ; **integer** *count*;

Procedure *dfs*(**vertex** v);

{∗ This is a recursive procedure. The call *dfs* (v) of the main program constructs a depth-first search tree T of G rooted at r; the recursive call *dfs*(w) constructs the subtree of T rooted at w. The depth-first search tree is constructed by placing the tree edges in the set T and labeling the vertices in the subtree rooted at vertex v in preorder numbering, starting with *count*. The procedure assigns ear labels to the edges of G while constructing the depth-first search tree. An edge that does not belong to any ear is given the label (∞, ∞). Initially, all vertices are unmarked. ∗}

vertex w;

'mark' v;

$preorder(v)$:= $count$; $count$:= $count + 1$; $low(v)$:= n; $ear(v)$:= (n, n);

for each vertex w adjacent to v $\rightarrow$

{∗ This **for** loop performs a depth-first search of each child of v in turn and assigns ear labels to the tree and nontree edges incident on vertices in the subtrees rooted at the children of v. ∗}

```
            if w is not marked →
                add (v, w) to T; parent(w) := v; dfs(w);
                if low(w) ≥ preorder(w) → ear(parent(w), w) := (∞, ∞)
            0.  | low(w) < preorder(w) → ear(parent(w), w) := ear(w)
                fi;
            1.  low(v) := min(low(v), low(w));
            2.  ear(v) := lexmin(ear(v), ear(w))
            | w is marked →
                if w ≠ parent(v) →
                3.  low(v) := min(low(v), preorder(w));
                4.  ear(w, v) := (preorder(w), preorder(v))
                5.  ear(v) := lexmin(ear(v), ear(w, v));
                fi
            fi
        rof

    end dfs;

    {* Main program. *}

    T := φ; count := 0; dfs(r);

    sort the ear labels of the edges in lexicographically nondecreasing order and
    relabel distinct labels (except label (∞, ∞)) in order as 1, 2, ...;

    relabel the nontree edge with label 1 as 0

end.
```

In the following we assume, for convenience, that the vertices are labeled by their preorder number.

Lemma 2.4 Tree edge $(parent(v),\ v)$ is a cutedge if and only if $low(v) \geq v$. If $low(v) < v$ for all $v \neq r$ then Algorithm 2.2 constructs an ear decomposition with each tree edge $(parent(v),\ v)$ contained in ear $P_{ear(v)}$.

Proof By the computation in steps 1 and 3 in Algorithm 2.2, $low(v)$ is the lowest numbered vertex w such that (x, w) is a nontree edge with x a descendant of v. Since nontree edges in a depth-first search tree go from a vertex to its ancestor, $low(v)$ is also the lowest numbered vertex in a fundamental cycle of a nontree edge incident on a descendant of v. If $low(v) \geq v$ then every nontree edge (y, z) incident on a descendant y of v has $z \geq v$. Hence tree edge $(parent(v), v)$ does not belong to any fundamental cycle and is a cutedge. Conversely, if $low(v) < v$ then there exists a nontree edge

$f = (x, low(v))$ with x a descendant of v. Hence tree edge $(parent(v),\ v)$ lies on the fundamental cycle of f and is not a cutedge.

Each nontree edge $(w, v), w < v$, is labeled (w, v) in step 4. We have $lca(w, v) = w$ since nontree edges in a depth first search go from a vertex v to an ancestor $w < v$. Hence the labels for the nontree edges are distinct and in nondecreasing order of their lca as required in step 2 of Algorithm 2.1.

By the computation in steps 2 and 5 in Algorithm 2.2, $ear(v)$ is set to be the lexicographic minimum among all nontree edges (u, w), with $u < w$ such that w is a descendant of v. In step 0 this label is assigned to tree edge $(parent(v),\ v)$. This is exactly the computation of step 3 of Algorithm 2.1 for assigning ear labels to tree edges. Hence by Lemma 2.3, Algorithm 2.2 constructs an ear decomposition for the input graph when it is 2-edge connected.[]

While Algorithm 2.2 is an ear decomposition algorithm, it also gives an open ear decomposition in case G is biconnected. We establish this in the next lemma.

Lemma 2.5 Algorithm 2.2 constructs an open ear decomposition if all of the following three conditions hold:

a) The root r has exactly one child c;

b) $low(c) = r$;

c) For all vertices v other than r and c, $low(v) < parent(v)$.

Further, G is biconnected if and only if a), b) and c) hold.

Proof We first prove that conditions a) through c) imply that Algorithm 2.2 constructs an open ear decomposition. We prove this by establishing that the ear containing each tree edge is open. This suffices to establish this part of the lemma since any ear that contains no tree edge consists of a single nontree edge, and such an ear is guaranteed to be open.

Consider tree edge $t = (parent(i),\ i)$. Let $low(i) = w$ and $ear(i) = q$.

Case 1: $q = 1$. Then t is contained in ear P_1 which is an open ear.

Case 2: $q > 1$. The ear containing edge t consists of the nontree edge with label q, call it (w, v), followed by part of the tree path from v to w (this was shown in the proof of Lemma 2.3). Let the part of the tree path from v to w that is contained in ear P_q extend from v to u, where u is a descendant of w and a proper ancestor of i (see figure 2.1). In order to show that ear P_q is open, it suffices to show that $u \neq w$.

Let (w, x) be the first tree edge on the path from w to i (figure 2.1). If w is not the root, then $low(x) < w$ (by condition c) and hence $ear(x) < q$. Thus edge (x, w) is not contained in ear P_q. Hence $u \geq x$, and since $x > w$, we are done. If w is the root then since $q > 1$, edge (w, x), which is equal to edge $(0, 1)$, has label 1, which is less than q. Hence $u \geq x$, and since $x > w$, we have $u > w$.

Hence the ear containing edge $(i, parent(i))$ is open. This concludes the proof of the statement that each tree edge is contained in an open ear. To complete the proof

Figure 2.1: Illustrating case 2 in the proof of Lemma 2.5

of the lemma we show that, if any one of conditions a) through c) is not satisfied, then G is not biconnected.

If condition a) is not satisfied, let c and d be two children of r with $c < d$. Then every path between c and d passes through r and hence r is a cutpoint and G is not biconnected.

If condition b) is not satisfied, then edge (r, c) is a cutedge and c is a cutpoint of G.

If condition c) does not hold, let v be a vertex for which it does not hold. The vertex v is neither the root nor the child of the root. If $low(v) > parent(v)$ then edge $(parent(v), v)$ is a cutedge (by proof of Lemma 2.4) and hence G is not biconnected. If $low(v) = parent(v) = w$ then any path between v and $parent(w)$ must pass through w. Hence w is a cutpoint of G.[]

Corollary to Lemma 2.5 Algorithm 2.2 constructs an open ear decomposition for a biconnected graph.

Lemma 2.5 does not hold if we use Algorithm 2.1 in place of Algorithm 2.2. Figure 2.2 gives two different ear decompositions that are obtained using Algorithm 2.1 on a given input graph with the same spanning tree but with two different edge orderings. Of these, one is an open ear decomposition while the other is not.

Figure 2.2: Examples of ear decompositions constructed by Algorithm 2.1

2.5 An Efficient Parallel Algorithm for Open Ear Decomposition

In the last section we noticed that Algorithm 2.1, when implemented using a depth-first search tree as the spanning tree for the input graph, serves as an algorithm to find an open ear decomposition of a biconnected graph; but if an arbitrary spanning tree is used, Algorithm 2.1 may no longer construct an open ear decomposition of a biconnected graph. Since no efficient parallel algorithm is known for finding a depth-first search tree in an undirected graph, we need to use a general spanning tree in an efficient parallel implementation of Algorithm 2.1.

Intuitively, the reason why a depth-first search tree is effective in finding an open ear decomposition is that all nontree edges go from a descendant to an ancestor. As a result the fundamental cycle of any nontree edge e contains only one base vertex v. Note that $lca(e) = parent(v)$. If the graph is biconnected, then there must be a path between v and some proper ancestor w (if it exists) of $lca(e)$ that avoids $lca(e)$. But this requires that edge $(parent(v), v)$ be contained in an ear that is incident on a proper ancestor of $lca(e)$.

When an arbitrary spanning tree is used in place of a depth-first search tree, the above property no longer need hold, and it is this that prevents Algorithm 2.1 from constructing a open ear decomposition for a biconnected graph. In order to address this, we will modify step 2 of Algorithm 2.1 to introduce some ordering among nontree edges with the same lca. The modified version of step 2 is given below.

Step $2'$.

$\{*$ Assign ear numbers for an open ear decomposition to nontree edges in T. $*\}$

pfor each vertex $v \in V - \{r\} \rightarrow$ compute $low(v)$ and 'mark' v if $low(v) <$

$parent(v)$ **rofp**;

a. construct an auxiliary multigraph $H = (V', E')$ with $V' = V - \{r\}$ and for each nontree edge e in G place an edge in E' between the base vertices of its fundamental cycle;

{$*$ In case e has only one base vertex u we place a self-loop at u. $*$}

pfor each connected component C of $H \rightarrow$

b. let a be any vertex in C and let b be the parent of a in T; $label(C) :=$ preorder number of b in T;

c. find a spanning tree S for C, root it at a 'marked' vertex if one exists, and number the vertices of S in preorder as $0, ..., k$;

d. label each tree edge $(parent(y),\ y)$ in S by the ordered pair $(label(C), y)$;

e. label the nontree edges in S (including multiple copies and self-loops) as $(label(C),\ k+1)$;

rofp;

pfor each nontree edge n in $G \rightarrow$ $label(n)$:= label of the edge in H that was placed in H by n **rofp**;

sort the labels of the nontree edges in G in lexicographically nondecreasing order and relabel them in order as $1, 2, ...$

end $2'$;

Lemma 2.6 Algorithm 2.1 with step 2 replaced by step $2'$ constructs an ear decomposition if G is two-edge connected.

Proof Let C be any connected component in H. The value of $label(C)$ computed in step b is the lca of the fundamental cycle of every nontree edge that places an edge in C in step a. Hence the labels assigned to nontree edges of G by step $2'$ continue to be nondecreasing in the lca of their fundamental cycle and hence by Lemma 2.3 the modified algorithm constructs an ear decomposition for G.[]

Lemma 2.7 Let C be a connected component in H.

a) If $label(C) \neq 0$ and C contains no marked vertex then G is not biconnected;

b) If $label(C) = 0$ and there is another connected component C' with $label(C') = 0$ then G is not biconnected.

Proof The proof is similar to the proof of the converse of Lemma 2.5 and is left as an exercise.[]

Theorem 2.1 Algorithm 2.1 with step 2 replaced by step $2'$ constructs an open ear decomposition of G if G is biconnected.

Proof By Lemma 2.6, P_1 is an open ear.

Let n be the nontree edge of T with label i, $i > 1$. Then by Lemma 2.3 we know that the edges in G with label i form a simple path p that is part of the fundamental cycle c of n. We will show that $p \neq c$ thereby establishing that P_i is an open ear.

Let $lca(n) = l$ and let a and b be the base vertices of the fundamental cycle of n. (Let $b = a$ if there is only one base vertex.) Then a and b belong to the same connected component C in H. Let $a \leq b$ in the numbering of step c. We will show that edge (l, a) must belong to an ear numbered lower than i.

Consider $ear(l, a)$. If a is a 'marked' vertex then edge (l, a) belongs to the fundamental cycle of a nontree edge whose lca is less than l and hence $ear(l, a) < i$. If a is not 'marked' then if a has a parent p in S, the spanning tree of C, then consider the nontree edge n' in G that introduced edge (a, p) in C. By the labeling scheme in steps d and e we have $label(n') < label(n)$. Further the fundamental cycle of n' contains the edge (a, l). Hence $ear(a, l) \leq label(n') < i$.

Finally if a is neither 'marked' nor has a parent in S (i.e., a is the root of S) then $C = 0$ by Lemma 2.7 and hence $ear(a, l) = 1 < i$.[]

Step $2'$ requires the computation the *low* value for the vertices, the computation of connected components, spanning trees, preorder numbering, and sorting. All of these computations can be performed in logarithmic time using a linear number of processors using well-known algorithms. Hence the over-all open ear decomposition algorithm (i.e., Algorithm 2.1 with step 2 replaced by step $2'$) has the same processor-time bounds.

3 Graph Triconnectivity

In this section we describe an algorithm for testing three-vertex connectivity (or triconnectivity) of a biconnected graph using an open ear decomposition of the graph. We then extend this algorithm to one that decomposes the biconnected graph into certain pieces called triconnected components. This material is from Miller & Ramachandran [MR87].

We start by presenting several definitions in Section 3.1. Since our algorithm is fairly complex, we give a high-level description of the approach in Section 3.2. In Section 3.3 we give the details of the triconnectivity algorithm and prove its correctness. In Section 3.4 we extend this algorithm to finding triconnected components.

In this section we only establish the correctness of the algorithm to test triconnectivity and find triconnected components using open ear decomposition. In Section 4 we describe implementations of the various steps in the algorithm that run in logarithmic time with a linear number of processors. At the end of the report we provide some pointers towards achieving optimal performance of the algorithm in logarithmic parallel time.

3.1 Further Graph-theoretic Definitions

We first need to add to the graph-theoretic definitions given in Section 2.2.

Let G be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Two ears are *parallel to each other* if they have the same endpoints; an ear P_i is a *parallel ear* if there exists another ear P_j such that P_i and P_j are parallel to each other.

An *st-numbering* of a graph G is a numbering of the n vertices of G from $s = 1$ to $t = n$, such that every vertex v (other than s and t) has adjacent vertices u, w with $u < v < w$. An *st-graph* is a directed acyclic graph $G = (V, E)$ with $(s, t) \in E$ such that every vertex in V lies on a path from s to t.

Let $P = \langle v_0, ..., v_{k-1}\rangle$ be a simple path. The path $P(v_i, v_j), 0 \le i, j \le k-1$ is the simple path connecting v_i and v_j in P, i.e., the path $\langle v_i, v_{i+1}, ..., v_j\rangle$, if $i \le j$ or the path $\langle v_j, v_{j+1}, ..., v_i\rangle$, if $j < i$. Analogously, $P[v_i, v_j]$ consists of the path (segments) obtained when the edges and internal vertices of $P(v_i, v_j)$ are deleted from P.

Given a noncyclic path $P = \langle v_0, ..., v_k\rangle$, the *innard of* P is the path $\langle v_1, ..., v_{k-1}\rangle$, i.e., the path obtained from P by deleting the first and last vertices.

Let $G = (V, E)$ be a biconnected graph, and let Q be a subgraph of G. We define the *bridges of* Q *in* G as follows: Let V' be the vertices in $G - Q$, and consider the partition of V' into classes such that two vertices are in the same class if and only if there is a path connecting them which does not use any vertex of Q. Each such class K defines a *nontrivial bridge* $B = (V_B, E_B)$ of Q, where B is the subgraph of G with $V_B = K \cup$ {vertices of Q that are connected by an edge to a vertex in K}, and E_B containing the edges of G incident on a vertex in K. The vertices of Q which are connected by an edge to a vertex in K are called the *attachments* of B on Q; the connecting edges are called the *attachment edges.* An edge (u, v) in $G - Q$, with both u and v in Q, is a *trivial bridge* of Q, with attachments u and v and attachment edge (u, v). The nontrivial and trivial bridges of Q together form the *bridges* of Q. The operation of *removing a bridge* B *of* Q *from* G is the removal from G of all edges and all nonattachment vertices of B.

Let $G = (V, E)$ be a graph and let $V' \subseteq V$ with the subgraph of G induced on V' being connected. The operation of *collapsing the vertices in* V' consists of replacing all vertices in V' by a single new vertex v, deleting all edges in G whose two endpoints are in V' and replacing each edge (x, y) with x in V' and y in $V - V'$ by an edge (v, y). In general this results in a multigraph even though G is not a multigraph.

Let $G = (V, E)$ be a biconnected graph, and let Q be a subgraph of G. The *bridge graph of* Q, $S = (V_S, E_S)$ is obtained from G by collapsing the nonattachment vertices in each nontrivial bridge of Q and by replacing each trivial bridge $b = (u, v)$ of Q by the two edges (x_b, u) and (x_b, v) where x_b is a new vertex introduced to represent the trivial bridge b. Note that in general the bridge graph is a multigraph.

Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. We will denote the bridge graph of ear P_i by C_i. The *anchor bridges of* P_i are the bridges of P_i in G that contain nonattachment vertices belonging to ears numbered lower than i. For any two vertices x, y on P_i, we denote by $V_i(x, y)$, the internal vertices of $P_i(x, y)$, i.e., the vertices in $P_i(x, y) - \{x, y\}$; we denote by $V_i[x, y]$, the vertices in $P_i[x, y] - \{x, y\}$ together with the nonattachment vertices in the anchor bridges of P_i. Figure 3.1 illustrates some of our definitions relating to bridges.

A *star* is a connected graph in which exactly one vertex has degree greater than 1. The unique vertex of a star that has degree greater than 1 is called its *center.*

Let P be a simple noncyclic path in a graph G. If each bridge of P in G contains exactly one vertex not on P, then we call G the *star graph* $G(P)$. Each bridge of $G(P)$ is a star and is called a *star of* $G(P)$. Note that, in a connected graph G, the bridge graph X of any simple noncyclic path in G is a star graph $X(P)$. For example, in figure 1, the bridge graph X of P_2 is a star graph $X(P_2)$. We will sometimes refer to a star graph $G(P)$ by G if the path P is clear from the context.

Let $G(P)$ be a star graph and let $P = \langle 0, 1, ..., k \rangle$. Given a star S of $G(P)$ with attachments $v_0 < v_1 < ... < v_r$ on P, we will call v_0 and v_r the *end attachments* of S and the remaining attachments the *internal attachments* of S; the vertex v_0 is the *leftmost attachment* of S, and the vertex v_r is its *rightmost attachment*; attachments v_i and v_{i+1} are *consecutive*, for $i = 0, \ldots, r-1$.

Two stars in a star graph $G(P)$ *interlace* if one of following two hold:

1) There exist four distinct vertices a, b, c, d in increasing order on P such that a and c are attachments of one of the stars and b and d are attachments of the other star; or

2) There are three distinct vertices on P that belong to both stars.

The operation of *coalescing* two stars S_j and S_k is the process of forming a single new star S_l from S_j and S_k by combining the centers of S_j and S_k, and deleting S_j and S_k. Given a star graph $G(P)$, a *coalesced graph* G_c of G is the graph obtained from G by repeatedly coalescing a pair of interlacing stars in the current star graph until no pair of stars interlace; a *partially coalesced graph* of G is any graph obtained from G by performing this repeated coalescing at least once.

A *planar embedding* of a graph G is a mapping of each vertex of G to a distinct point on the plane and each edge of G to a curve connecting its endpoints such that no two edges intersect. A *face* of a planar embedding is a maximal region of the plane that is bounded by edges of the planar embedding. The *outer face* of a planar embedding is the face with unbounded area. An *inner face* of a planar embedding is a face with finite area.

Let $G(P)$ be a star graph in which no pair of stars interlace. If $G(P)$ contains no star that has attachments to the endpoints x and y of P, then add a virtual star X to $G(P)$ with attachments to x and y. The *star embedding* $G^*(P)$ *of* $G(P)$ is the planar embedding of (the possibly augmented) $G(P)$ with P on the outer face. From some well-known results in planarity, it can be established that a star graph $G(P)$ has a planar embedding with P on the outer face if and only if no pair of stars interlace. We give some further definitions on planar combinatorial embeddings in Section 4.2.3.

G with open ear decomposition $D = [P_0, P_1, P_2,\ P_3,\ P_4, P_5]$; $P_0 = \langle a, b \rangle$,
$P_1 = \langle b, c, d, e, a \rangle$, $P_2 = \langle c, g, f, e \rangle$, $P_3 = \langle d, f \rangle$, $P_4 = \langle g, h, f \rangle$, $P_5 = \langle c, i, e \rangle$.

Bridges of P_2.

Bridge graph X of P_2. Anchor bridges are B_1 and B_3

Figure 3.1: Illustrating the definitions

Let $G(P)$ be a star graph with a star embedding $G^*(P)$. Let B and B' be two stars in $G(P)$. Then B is the *parent-star* of B' and B' is a *child-star* of B if there is a face in the star embedding $G^*(P)$ that contains the end attachment edges of B' as well as an attachment edge of B on either side of the end attachments of B'.

Let G be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Let $B_1, ..., B_l$ be the anchor bridges of ear P_i. The *ear graph of* P_i, denoted by $G_i(P_i)$, is the graph obtained from the bridge graph of P_i by

a) Coalescing all stars corresponding to anchor bridges;

b) Removing any multiple two-attachment bridges with the endpoints of the ear as attachments; and

c) Replacing all multiple edges by a single copy.

We will call the star obtained by coalescing all anchor bridges, the *anchoring star* of $G_i(P_i)$.

We conclude our list of definitions by defining the *triconnected components* of a biconnected multigraph (see, e.g., [Tu66, HT72]).

A pair of vertices a, b in a multigraph $G = (V, E)$ is a separating pair if and only if there are two nontrivial bridges, or at least three bridges, one of which is nontrivial, of $\{a, b\}$ in G. A biconnected graph with at least four vertices is triconnected if it has no separating pair. The pair a, b is a *nontrivial* separating pair if there are two nontrivial bridges of a, b in G. These definitions apply to a (simple) graph as well; in this case, all separating pairs are nontrivial. By Menger's theorem, a graph is triconnected if and only if it is biconnected and has at most 3 vertices or there are 3 vertex-disjoint paths between every pair of distinct vertices.

Let $\{a, b\}$ be a separating pair for a biconnected multigraph $G = (V, E)$. For any bridge X of $\{a, b\}$, let $\bar{X}$ be the induced subgraph of G on $(V - V(X)) \cup \{a, b\}$. Let B be a bridge of $\{a, b\}$ such that $|E(B)| \geq 2, |E(\bar{B})| \geq 2$ and either B or $\bar{B}$ is biconnected. We can apply a *Tutte split* $s(a, b, i)$ to G by forming G_1 and G_2 from G, where G_1 is $B \cup \{(a, b, i)\}$ and G_2 is $\bar{B} \cup \{(a, b, i)\}$. Note that we consider G_1 and G_2 to be two separate graphs. Thus it should cause no confusion that there are two edges labeled (a, b, i) since one of these edges is in G_1 and the other is in G_2. The graphs G_1 and G_2 are called *split graphs of G with respect to a, b*. The *Tutte components* of G are obtained by successively applying a Tutte split to split graphs until no Tutte split is possible. Every Tutte component is one of three types: i) a triconnected simple graph; ii) a simple cycle (a *polygon* or iii) a pair of vertices with at least three edges between them (a *bond* the Tutte components of a biconnected multigraph G are the unique *triconnected components* of G.

3.2 Brief Overview of Results

In this section we give a high-level description of the results leading to our triconnectivity algorithm. Given a biconnected graph, our algorithm finds all separating pairs in the graph. The input graph is triconnected if and only if the algorithm finds no separating pair in the graph.

In Section 3.3 we show that if x, y is a separating pair in a biconnected graph G with an open ear decomposition D, then there exists an ear P_i in D that contains x and y as nonadjacent vertices, and further, every bridge of P_i has an empty intersection with either $V_i(x, y)$ or $V_i[x, y]$. This is the basic property that we use in our algorithm.

We further show that the above property is not altered by the operation of coalescing interlacing stars in the bridge graph $C_i(P_i)$ and thus applies to the ear graph of P_i as well as its coalesced graph. Finally we show that separating pairs satisfying the basic property with respect to P_i are simply those pairs of nonadjacent vertices on P_i that lie on a common face in the star embedding of this coalesced graph.

The above results lead to the following high-level algorithm for finding separating pairs in a biconnected graph G: Find an open ear decomposition D for G and for each nontrivial ear P_i in D, form the coalesced graph of its ear graph and extract separating pairs from its star embedding.

In Section 3.4 we build on the above results to give an efficient parallel algorithm to find the triconnected components of a graph. This algorithm finds the triconnected components using Tutte splits in contrast to the earlier algorithm based on depth first search [HT72], which used certain other types of splits that required a clean-up phase at the end of the algorithm.

The definition of triconnected components given in Section 3.1 may appear contrived at first, but in reality it decomposes a biconnected graph into substructures that preserve the triconnected structure of G. In particular, questions relating to graph planarity and isomorphism between a pair of graphs can be mapped onto related questions regarding the triconnected components. Thus the problem of finding the triconnected components of a graph is an important one.

3.3 The Triconnectivity Algorithm

Lemma 3.1 Let $D = [P_0, ..., P_{r-1}]$ be an open ear decomposition of a biconnected graph G and let x and y be the endpoints of ear P_i. Then every anchor bridge of P_i has attachments on x and y.

Proof Let B be an anchor bridge of P_i and let $H = \cup_{j=0}^{i-1} P_j$. By definition, the nonattachment vertices in B are the vertices in a connected component C of $G - \{P_i\}$ that contains a vertex in $H - \{x, y\}$.

The graph $(H - \{x, y\}) \cap P_i$ is empty since none of the internal vertices of P_i are contained in ears numbered lower than i. Hence C must contain all vertices in one or more connected component(s) of $H - \{x, y\}$. Let D be one such connected component contained in C. Since H has an open ear decomposition, it is biconnected by Lemma 2.2. Hence D contains vertices adjacent to x and y in H, since otherwise x or y would be a cutpoint of H. But this implies that C contains vertices adjacent to x and y in $G - \{P_i\}$, i.e., bridge B of P_i has attachments on x and y.[]

Figure 3.2: Case 1 in the proof of Lemma 3.2

Lemma 3.2 Let $G = (V, E)$ be a biconnected undirected graph for which vertices x and y form a separating pair. Let $D = [P_0, ..., P_{r-1}]$ be an open ear decomposition for G. Then there exists a nontrivial ear P_i in D that contains x and y as nonadjacent vertices, such that every path from a vertex in $V_i(x, y)$ to a vertex in $V_i[x, y]$ in G passes through either x or y.

Proof Since x and y form a separating pair, the subgraph of G induced by $V - \{x, y\}$ contains at least two connected components. Let X_1 and X_2 be two such connected components.

Case 1: The ear P_1 contains no vertex in X_2 (see figure 3.2):

Consider the lowest-numbered ear, P_i, that contains a vertex v in X_2. Since the endpoints of P_i are distinct and must be contained in ears numbered lower than i, P_i must contain x and y. Further, all vertices in $V_i(x, y)$ lie in X_2, and none of the vertices in $V_i[x, y]$ lie in X_2. Hence every path from a vertex in $V_i(x, y)$ to a vertex in $V_i[x, y]$ in G passes through either x or y. Further, x and y are not adjacent on P_i since v lies between x and y.

Case 2: P_1 contains a vertex in X_2:

If P_1 contains no vertex in X_1, then case 1 applies to X_1. Otherwise P_1 contains at least one vertex from X_1, and one vertex from X_2. But then, since $P_0 \cup P_1$ is a simple cycle, and P_1 contains both vertices in P_0, we have the result that P_1 must contain x and y. Hence, by the argument of Case 1, every path from a vertex in $V_1(x, y)$ to a vertex in $V_1[x, y]$ must contain either x or y, and x and y are not adjacent on P_1.[]

We will say that a separating pair x, y *separates* ear P_i if x and y are nonadjacent vertices on P_i, and the vertices in $V_i(x, y)$ are disconnected from the vertices in $V_i[x, y]$ in the subgraph of G induced by $V - \{x, y\}$. By Lemma 3.2, every separating pair in G separates some nontrivial ear. (Note that a separating pair may separate more than one nontrivial ear; for instance, in the graph G in figure 3.1, the pair c, e is a pair separating ears P_1 and P_5,—note that c, e does not separate P_2.)

Lemma 3.3 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Let ear P_i contain x and y as nonadjacent vertices. Then x, y separates P_i if and only if every bridge of P_i has an empty intersection with either $V_i(x, y)$ or $V_i[x, y]$.

Proof Let every bridge of P_i have an empty intersection with either $V_i(x, y)$ or $V_i[x, y]$ and suppose x, y does not separate ear P_i. Hence, there exists a path $P = \langle a, w_1, ..., w_l, b\rangle$ in G, with a in $V_i(x, y)$ and b in $V_i[x, y]$, that avoids both x and y. This implies that there is a subpath P' of P with $P' = \langle w_r, ..., w_s\rangle$ such that w_r is in $V_i(x, y)$, w_s is in $V_i[x, y]$, and none of the intermediate w_k lie on P_i. Hence there is a bridge B of P_i containing w_r and w_s, i.e., B has a nonempty intersection with both $V_i(x, y)$ and $V_i[x, y]$, which is not possible by assumption. Hence x, y must separate ear P_i.

Conversely suppose B is a bridge of P_i containing a vertex a in $V_i(x, y)$ and a vertex b in $V_i[x, y]$. Then we have a path from a vertex in $V_i(x, y)$ to a vertex in $V_i[x, y]$ that avoids both x and y. Hence x, y does not separate P_i.[]

Corollary to Lemma 3.3 Let x and y be the endpoints of a nontrivial ear P_i in an open ear decomposition D of a graph G. Then x, y separates P_i if and only if no anchor bridge of P_i has an attachment in $V_i(x, y)$.

Proof Let x, y separate P_i. By Lemma 3.3, every bridge of P_i has an empty intersection with either $V_i(x, y)$ or $V_i[x, y]$. Since any anchor bridge of P_i has a nonempty intersection with $V_i[x, y]$, every anchor bridge must have an empty intersection with $V_i(x, y)$. Hence no anchor bridge can have an attachment in $V_i(x, y)$.

Conversely, suppose no anchor bridge of P_i has an attachment in $V_i(x, y)$. Then every anchor bridge has an empty intersection with $V_i(x, y)$. Since x and y are endpoints of P_i, every nonanchor bridge has an empty intersection with $V_i[x, y]$. Hence by Lemma 3.3, x, y separates P_i.[]

We will call a pair of vertices x, y on an ear P_i a *candidate pair for* P_i if x, y is a pair separating P_i or (x, y) is an edge in P_i or x and y are endpoints of P_i. Clearly, if we can determine the set of candidate pairs for P_i, we can extract from it the pairs separating P_i by deleting pairs that are endpoints of an edge in P_i, and checking if the endpoints of P_i form a pair separating P_i using the criterion in the above Corollary.

More generally, let $G(P)$ be a star graph. A pair of nonadjacent vertices x, y on P will be called a *pair separating* P if the vertices in $P(x, y) - \{x, y\}$ are separated from the vertices in $P[x, y] - \{x, y\}$ when x and y are deleted from G. A pair of vertices x, y on P will be called a *candidate pair* for P in G if x, y is a pair separating P, or x and y are endpoints of P, or (x, y) is an edge in P.

The proof of the following claim is similar to the proof of Lemma 3.3 and is omitted.

Claim 3.1 Let $G(P)$ be a star graph. A pair x, y separates P in $G(P)$ if and only if every bridge of P in $G(P)$ has an empty intersection with either $P(x, y) - \{x, y\}$ or $P[x, y] - \{x, y\}$.

We now relate candidate pairs for P_i in G with candidate pairs for P_i in its bridge graph $C_i(P_i)$.

Observation 3.1 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Then x, y is a candidate pair for P_i in G if and only if it is a candidate pair for P_i in the bridge graph $C_i(P_i)$.

Proof If (x, y) is an edge in P_i or if x and y are endpoints of P_i, then x, y is a candidate pair for P_i in both G and $C_i(P_i)$. So in the following we assume that x, y separates P_i and x and y are not both endpoints of P_i.

Let x, y separate P_i in G. By Lemma 3.3 every bridge of P_i in G has an empty intersection either with $V_i(x, y)$ – and hence with $P_i(x, y) - \{x, y\}$ – or with $V_i[x, y]$ – and hence with $P_i[x, y] - \{x, y\}$. By construction this implies that every bridge of P_i in $C_i(P_i)$ has an empty intersection either with $P_i(x, y) - \{x, y\}$ or with $P_i[x, y] - \{x, y\}$. Hence by Claim 3.1, x, y separates P_i in $C_i(P_i)$.

Conversely, let x, y separate P_i in $C_i(P_i)$. By Claim 3.1, every bridge of P_i in $C_i(P_i)$ has an empty intersection either with $P_i(x, y) - \{x, y\}$ or with $P_i[x, y] - \{x, y\}$. Let $B_1, ..., B_k$ be the bridges of P_i in $C_i(P_i)$ corresponding to the anchor bridges of P_i in G. By Lemma 3.1, each B_j has attachments to the two endpoints e and f of P_i and by assumption either e or f is distinct from x and y. Assume without loss of generality that e is different from x and y. The vertex e is in $P_i[x, y] - \{x, y\}$ and each $B_j, j = 1, ..., k$ has an attachment on e. Hence each B_j has a nonempty intersection with $P_i[x, y] - \{x, y\}$ and therefore must have an empty intersection with $P_i(x, y) - \{x, y\}$.

The above implies that every anchor bridge of P_i in G has an empty intersection with $V_i(x, y)$ and every nonanchor bridge has an empty intersection either with $V_i(x, y)$ or with $V_i[x, y]$. Hence, by Lemma 3.3, x, y separates P_i in G.[]

By the above Observation we can work with the bridge graph of each ear in order to find the candidate pairs for that ear in G. We now develop results that will lead to an efficient algorithm to find candidate pairs in a star graph.

Lemma 3.4 Let $G(P)$ be a star graph with stars $S_1, ..., S_k$. For $j = 1, ..., k$ let H_j be the subgraph of G consisting of $P \cup S_j$ and let $H_j{}^*$ be the star embedding of H_j. Then a pair of vertices x, y on P is a candidate pair for P if and only if either x and y are the endpoints of P or x and y lie on a common inner face in each $H_j{}^*, j = 1, ..., k$.

Proof Let x, y be a candidate pair for P. If x and y are endpoints of P then the result follows immediately. If (x, y) is an edge on P then x and y must lie on a common inner face in each $H_j{}^*$. Otherwise, by Claim 3.1, each S_j has an empty intersection with either $P(x, y) - \{x, y\}$ or $P[x, y] - \{x, y\}$.

If S_j has an empty intersection with $P[x, y] - \{x, y\}$ then x and y belong to the unique inner face of H_j^* that contains the endpoints of P. If S_j has an empty intersection with $P(x, y) - \{x, y\}$, let $\langle a_1, ..., a_l \rangle$ be the attachments of S_j on P in the order that they are encountered on P from one endpoint of P to the other. The vertices x and y must lie between a_p and a_{p+1}, for some $1 \leq p < k$. Then x and y lie on the unique inner face of $H_j{}^*$ containing a_p and a_{p+1}.

If x, y is not a candidate pair for P, then by Claim 3.1 there exists a star S_j with consecutive attachments a, b, with a in $P[x,y]-\{x,y\}$ and b in $P(x,y)-\{x,y\}$. Then, one of x and y, say x, lies in $P(a,b)-\{a,b\}$ and the other, y, lies in $P[a,b]-\{a,b\}$. Then x lies on the unique inner face containing a and b in H_j^* and y does not lies on this face.[]

Corollary to Lemma 3.4 If G^* is the star embedding of $G(P)$, then a pair of vertices x, y on P is a candidate pair for P if and only if either x and y are the endpoints of P or x and y lie on a common inner face in G^*.

In general, this corollary may not apply, because $G(P)$ need not be planar. We now introduce the star coalescing property: namely, we establish that if we enforce the planarity required in the corollary by forming a coalesced graph G_c of $G(P)$ then the corollary applies to G_c.

The coalesced graph $G_c(P)$ of a star graph $G(P)$ is unique (exercise 3). Hence in the following we refer to G_c as 'the' coalesced graph of G (rather than 'any' coalesced graph of G).

Theorem 3.1 Let $G(P)$ be a star graph and let $G_1(P)$ be obtained from $G(P)$ by coalescing a pair of interlacing stars S and T. Then a pair x, y on P is a candidate pair for $G(P)$ if and only if it is a candidate pair for $G_1(P)$.
Proof Let R be the star in $G_1(P)$ formed by coalescing S and T.

If (x, y) is an edge on P or if x and y are endpoints of P then x, y is a candidate pair for both $G(P)$ and $G_1(P)$.

Let x, y separate P in $G(P)$. Hence S and T have an empty intersection with either $P(x,y)-\{x,y\}$ or $P[x,y]-\{x,y\}$. Since S and T interlace, either both have empty intersection with $P(x,y)-\{x,y\}$ or both have empty intersection with $P[x,y]-\{x,y\}$. Hence R, which contains the union of the attachments of S and T, must have an empty intersection with either $P(x,y)-\{x,y\}$ or with $P[x,y]-\{x,y\}$. Hence by Claim 3.1, x, y separates P in $G_1(P)$.

Conversely suppose x, y separates P in $G_1(P)$ and let R have an empty intersection with $P(x,y)-\{x,y\}$ ($P[x,y]-\{x,y\}$). Then both S and T have an empty intersection with $P(x,y)-\{x,y\}$ ($P[x,y]-\{x,y\}$) and hence x, y separates P in $G(P)$ by Claim 3.1.[]

Corollary to Theorem 3.1 Let $G(P)$ be a star graph.

a) Let $G'(P)$ be any partially coalesced graph of $G(P)$. Then x, y is a candidate pair for $G(P)$ if and only if it is a candidate pair for $G'(P)$.

b) A pair x, y is a candidate pair for $G(P)$ if and only if it is a candidate pair for the coalesced graph $G_c(P)$.

Let $G(P)$ be a star graph and let $G_c(P)$ be its coalesced graph. Since no pair of bridges of P interlace in $G_c(P)$, Lemma 3.4 and its Corollary apply to this graph. Let us refer to the set of vertices on P that lie on a common inner face in ${G_c}^*$ listed in the order they appear on P as a *candidate list for* P. A pair of vertices is a candidate pair for P if and only if it lies in a candidate list for P. A candidate list S for ear P is a *nontrivial candidate list* if it contains a pair separating P.

Let G be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Since every separating pair for G is a candidate pair for some nontrivial ear P_i (Lemma 3.2), any algorithm that determines the candidate lists for all nontrivial ears is an algorithm that finds all separating pairs for a graph. By the results we have proved above, we can find all candidate lists in G by forming the bridge graph for each nontrivial ear, and then extracting the nontrivial candidate lists from the coalesced graph of the bridge graph.

In order to obtain an efficient implementation of this algorithm, we will not use the bridge graph of each ear, but instead the closely related ear graph which we defined in Section 3.1.

Lemma 3.5 A pair of vertices x, y separates ear P_i in G if and only if it separates P_i in the ear graph $G_i(P_i)$.

Proof By Claim 3.1, x, y separates ear P_i in G if and only if it separates P_i in the bridge graph $C_i(P_i)$.

Now consider the ear graph $G_i(P_i)$. The ear graph $G_i(P_i)$ is obtained from the bridge graph $C_i(P_i)$ by coalescing all anchor bridges, deleting multiple two-attachment bridges with the endpoints of the ear as attachments, and deleting all multiple edges by a single copy.

Deleting a star with attachments only to the endpoints of an ear can neither create nor destroy candidate pairs. Let $C_i'(P_i) = C_i(P_i) - \{$2-attachment bridges with endpoints of P_i as attachments$\}$.

By Lemma 3.1, every anchor bridge of P_i has the two endpoints of P_i as attachments, and hence every pair of anchor bridges with an internal attachment on P_i must interlace. Hence $G_i(P_i)$ is the graph derived from $C_i'(P_i)$ by coalescing some interlacing stars. The lemma now follows from the Corollary to Theorem 3.1.[]

Lemma 3.6 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$, and let $|V| = n$ and $|E| = m$. Then the total size of the ear graphs of all nontrivial ears in D is $O(m)$.

Proof Each ear graph consists of a nontrivial ear P_i together with a collection of stars on P_i. The size of all of the P_i is $O(m)$. So we only need to bound the size of all of the stars in all of the ear graphs.

Consider an edge (u, v) in G. This edge appears as an internal attachment edge in at most two ear graphs: once for the ear $P_{ear(u)}$ and once for ear $P_{ear(v)}$. Thus the number of internal attachment edges in all of the stars is no more than $2m$.

We now bound the number of attachment edges to endpoints of ears. Since we delete all stars with only the endpoints of an ear as attachments, every star in an ear graph $G_i(P_i)$ with an attachment to an endpoint of P_i also has an internal attachment in P_i. A star can contain at most two attachments to endpoints of an ear. Hence for each star that contains attachments to endpoints of its ear, we charge these attachments to an internal attachment. Since the number of internal attachment edges is no more than $2m$, the number of attachment edges to endpoints of ears is no more than $4m$. Hence the total size of all of the ear graphs is $O(m)$.[]

The above results establish the validity of the following algorithm to find the nontrivial candidate lists in a biconnected graph.

Algorithm 3.1: Finding the Nontrivial Candidate Lists
Input: A biconnected graph $G = (V, E)$.
Output: The candidate lists for G.

integer j; **vertex** u, v;

1. find an open ear decomposition $D = [P_0, ..., P_{r-1}]$ for G;

pfor each nontrivial ear $P_j \rightarrow$

2. construct the ear graph $G_j(P_j)$;
3. coalesce all interlacing stars on $G_j(P_j)$ to form the coalesced graph G_{j_c};
4. construct the star embedding of $G^*_{j_c}$ of G_{j_c}, and identify each list of vertices on P_j on a common inner face in this embedding as a candidate list;

   let u and v be the endpoints of P_j;

   if $[u, v]$ is a candidate list for P_j and the anchoring star of P_j has an internal attachment on P_j $\rightarrow$ delete candidate list $[u, v]$ **fi**;

   delete any candidate list for P_j that contains only the two endpoints of an edge in P_j

rofp

end.

In Section 2.5 we described a logarithmic time parallel algorithm with a linear number of processors on a CRCW PRAM for step 1 of Algorithm 3.1. In Section 4, we give algorithms with similar processor-time bounds to perform steps 2, 3 and 4 in parallel for all nontrivial ears. Clearly the remaining steps in the **pfor** loop are trivial to implement. Hence Algorithm 3.1 can be made to run in logarithmic time with a linear number of processors. However, before proceeding to an efficient implementation of Algorithm 3.1, we show in Section 3.4, how to obtain the triconnected components of a biconnected graph, given the nontrivial candidate lists.

3.4 Finding Triconnected Components

In this section we define a special type of split, called the *ear split* in a biconnected graph with an open ear decomposition. This split has the desirable property that the original open ear decomposition decomposes in a natural way into two open ear decompositions, one for each split graph. This also leads to a natural algorithm for finding triconnected components based on applying certain types of ear splits successively.

We also consider some issues that arise in a parallel implementation of the above algorithm. The obvious approach would be to perform all of the ear splits in parallel. However, this leads to complications when a vertex is shared by several Tutte pairs. We analyze some of the properties of ear splits in this section and we present a method for performing all of the relevant ear splits on a single ear. This method runs in logarithmic time with a number of processors linear in the size of the bridge graph of the ear. In Section 4.3 we apply this method to the 'local replacement graph' which is defined in Section 4.1 to obtain a logarithmic time algorithm using a linear number of processors to find the triconnected components of the input graph.

We start by defining a special type of split, called an *ear split,* on a biconnected graph G with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Let a, b be a pair separating ear P_i. Let $B_0, ..., B_k$ be the bridges of P_i with an attachment in $V_i(a, b)$, and let $T_i(a, b) = (\cup_{j=0}^{k} B_j) \cup P_i(a, b)$. It is easy to see that $T_i(a, b)$ is a bridge of a, b. Then the *ear split* $e(a, b, i)$ consists of forming the *upper split graph* $G_1 = T_i(a, b) \cup \{(a, b, i)\}$ and the *lower split graph* $G_2 = \bar{T}_i(a, b) \cup \{(a, b, i)\}$. Note that the ear split $e(a, b, i)$ is a Tutte split if one of $G_1 - \{(a, b, i)\}$ or $G_2 - \{(a, b, i)\}$ is biconnected.

Let S be a nontrivial candidate list for ear P_i. A pair u, v in S is an *adjacent separating pair for* P_i if S contains no vertex in $V_i(u, v)$. The pair u, v is a *nonvacuous adjacent separating pair for* P_i if u, v is an adjacent separating pair and there is a bridge of P_i with an attachment on $V_i(u, v)$. A pair a, b in S is an *extremal separating pair for* P_i if $|S| \geq 3$ and S contains no vertex in $V_i[a, b]$. We will refer to a nonvacuous adjacent or extremal separating pair as a *Tutte pair.*

We now prove the following theorem.

Theorem 3.2 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$. Let a, b be an adjacent (extremal) separating pair for P_i in G, and let G_1 and G_2 be, respectively, the upper and lower split graphs obtained by the ear split $e(a, b, i)$. Then,

a) $G_1 - \{(a, b, i)\}$ $(G_2 - \{(a, b, i)\})$ is biconnected.

b) The ear decomposition D_1 induced by D on G_1 by replacing P_i by the simple cycle formed by $P_i(a, b)$ followed by the newly added edge (b, a, i) is a valid open ear decomposition for G_1; likewise, the ear decomposition D_2 induced by D on G_2 by replacing $P_i(a, b)$ by the newly added edge (a, b, i) is a valid open ear decomposition for G_2.

c) Let c, d be a pair separating some $P_j, 0 \leq j \leq r-1$ in G. If $\{c, d\} \neq \{a, b\}$ or $i \neq j$ then c and d lie in one of G_1 or G_2, and c, d is a separating pair for P_j in the split graph in which $P_j, c,$ and d lie.

d) Every separating pair in G_1 or in G_2 is a separating pair in G.

Proof
a) Let a, b be an adjacent separating pair for P_i. If $G_1 - \{(a,b,i)\}$ is not biconnected then let c be a cutpoint in the graph. The vertex c cannot lie on $P_i(a,b)$ since this would imply that it is part of the candidate list for which a, b is an adjacent separating pair. But c cannot lie on a bridge of $P_i(a,b)$ since then c would be a cutpoint of G and this would imply that G is not biconnected.

Similarly $G_2 - \{(a,b,i)\}$ is biconnected if a, b is an extremal separating pair.
b) We establish by induction on ear number j, for $j \geq i$, that the graph $P_{0,j} = \cup_{k=0}^{j} P_k$ satisfies the property in part b) of the Theorem. The details are straightforward and are omitted.
c and d) If $i \neq j$ let P_j lie in G_k (where k =1 or 2). We note that the ear graph of P_j in G_k is the same as the ear graph of P_j in G. Hence c, d is a pair separating P_j in G if and only if it is a pair separating P_j in G_k.

If $i = j$ we note that in G_1 the bridges of P_i are precisely those bridges of P_i in G that have attachments on an internal vertex of $P_i(a,b)$. Hence if c and d lie on $P_i(a,b)$ then c, d separates P_i in G if and only if it separates $P_i(a,b)$ in G_1. An analogous argument holds for G_2 in the case when c and d lie on $P_i[a,b]$.[]

We now present the algorithm for finding triconnected components.

Algorithm 3.2: Finding Triconnected Components
Input: A biconnected graph $G = (V, E)$ with an open ear decomposition $D = [P_0, ..., P_{r-1}]$, and the nontrivial candidate lists for each ear.
Output: The triconnected components of G.

> **vertex** u, v; **integer** i;
>
> **pfor** each nontrivial candidate list S in each nontrivial ear $P_i \rightarrow$
>
> > **pfor** each nonvacuous adjacent separating pair u, v in $S \rightarrow$
> >
> > > form the upper split graph G_1 for the ear split $e(u,v,i)$ and replace G by the lower split graph G_2 for the ear split $e(u,v,i)$;
> > >
> > > replace D by the open ear decomposition D_2 for the lower split graph G_2 and form the open ear decomposition D_1 for the upper split graph G_1 as in part b) of Theorem 3.2
> >
> > **rofp**;
> >
> > **if** $|S| > 2 \rightarrow$
> >
> > > form the upper split graph G_1 and replace G by the lower split graph G_2 for the extremal separating pair u, v in S;
> > >
> > > form the open ear decompositions D_1 and D_2 as in Theorem 3.2 and replace D by D_2. (if $i = 1$ and u and v are endpoints of ear P_1 then perform this ear split only if there are at least two edges between u and v)

fi

rofp;

split off multiple edges in the remaining split graphs to form the bonds

end.

Lemma 3.7 Algorithm 3.2 generates the Tutte components of G.
Proof By Theorem 3.2, each split performed in Algorithm 3.2 is a Tutte split, and at termination there is no separating pair in any of the generated graphs.[]

For an efficient parallel implementation of Algorithm 3.2 we need a good method to perform all of the Tutte splits in the algorithm in parallel. This is quite simple if all of the Tutte pairs are disjoint. However, for the general case when the Tutte pairs are not necessarily disjoint, we need to specify a method to process the splits in parallel without causing conflicts between different splits that share a vertex in their Tutte pairs. In the rest of this section we develop a method to perform in parallel all of the splits on Tutte pairs in a single ear. This method is not necessarily efficient. However, it will be used in a general algorithm described in Section 4.3 that performs the splits corresponding to Tutte pairs in all ears in logarithmic time with a linear number of processors.

We start by associating a triconnected component with each ear split corresponding to a Tutte pair. Let $e(a,b,i)$ be such a split. Then by definition $T_i(a,b) \cup \{(a,b,i)\}$ is the upper split graph associated with the ear split $e(a,b,i)$. The *triconnected component of the ear split* $e(a,b,i)$, denoted by $TC(a,b,i)$, is $T_i(a,b) \cup \{(a,b,i)\}$ with the following modifications: Call a pair c,d separating an ear P_j in $T_i(a,b)$ a *maximal pair for* $T_i(a,b)$ if there is no e,f in $T_i(a,b)$ such that e,f separates some ear P_k in $T_i(a,b)$ and c,d is in $T_k(e,f)$. In $T_i(a,b) \cup \{(a,b,i)\}$ replace $T_j(c,d)$ together with all two-attachment bridges with attachments at c and d, for each maximal pair c,d of $T_i(a,b)$, by the edge (c,d,j) to obtain $TC(a,b,i)$. We denote by $TC(0,0,0)$, the unique triconnected component that contains P_0.
Lemma 3.8 $TC(a,b,i)$ is a triconnected component of G.
Proof Each split of $T_i(a,b)$ in the above definition is a valid Tutte split, and the final resulting graph contains no unprocessed separating pair. Hence $TC(a,b,i)$ is a valid triconnected component of G.[]
Lemma 3.9 Every triconnected component of G is $TC(a,b,i)$ for some unique triplet (a,b,i).
Proof Straightforward.[]

We note that if a,b is an extremal pair separating P_i then $TC(a,b,i)$ is a polygon and if a,b is a nonvacuous adjacent pair separating P_i then $TC(a,b,i)$ is a simple triconnected graph.

Let $G=(V,E)$ be a biconnected graph with an open ear decomposition $D=[P_0,P_1,...,P_{r-1}]$. Let $C_i(P_i)$ be the bridge graph of P_i and let $D_i(P_i)$ be the coalesced graph of C_i. Note that D_i is closely related to $G_{i_c}(P_i)$, the coalesced graph of the ear graph of P_i in G, but is not exactly the same since D_i retains multiple attachment edges as well as multiple two-attachment bridges. (Note also that the sum of the sizes of the D_i over all nontrivial ears could be superlinear in the size of G.)

The proofs of the following two lemmas are left as exercises.

Lemma 3.10 Algorithm 3.1 with G_{i_c} replaced by D_i will output the nontrivial candidate lists of G.

Lemma 3.11 Let a, b be a nonvacuous adjacent separating pair for P_i in G and let (x, y) be an edge, not in P_i, which is incident on a vertex y on P_i. Then

a) The edge (x, y) is in $T_i(a, b)$ if and only if it is in a star of D_i with an attachment on an internal vertex in $P_i(a, b)$;

b) D_i contains at most one star B with attachments on a, b, and an internal vertex in $P_i(a, b)$, and if edge (x, y) is in $TC(a, b, i)$ then it lies in B.

We now give a lemma about two-attachment bridges.

Lemma 3.12 Let B be a two-attachment bridge of P_i in D_i with attachments a and b. Then

a) If the span $[a, b]$ is degenerate (i.e., (a, b) is an edge in P_i) or if there is a bridge B' of P_i with attachments on a and b and at least one other vertex, then the graph $D_i - B$ defines the same set of polygons and simple triconnected components $TC(x, y, i)$, for i fixed, as $D_i(P_i)$.

b) If part a) does not hold then $\{a, b\}$ is an extremal pair separating P_i as well as an adjacent pair separating P_i.

Proof Let P_j be the lowest-numbered ear in B. Then $j > i$ and a and b are endpoints of P_j. Hence the ear split $e(a, b, j)$ separates B from P_i, and thus B is not part of $TC(x, y, i)$ for any pair $\{x, y\}$ separating P_i. So a two-attachment bridge of P_i in D_i is never part of a triconnected component associated with a pair separating P_i, though it may define some adjacent and extremal separating pairs as in case b) of the lemma.

We now prove parts a) and b) of the lemma.

Part a): Suppose span $[a, b]$ is degenerate. Then the triconnected component associated with split $e(a, b, i)$ is the single edge (a, b), which is a bond. Otherwise, if there is a bridge B' with attachments on a, b and at least one other vertex v, then the triconnected component associated with split $e(a, b, i)$ contains a portion of P_i between a and b, together with B' if v is in the interval (a, b) and is a polygon if v is not in $[a, b]$. Both of these situations can be inferred without the presence of B. Note that it is not possible for B' to have an attachment v in the interval (a, b) and another attachment w that is not in $[a, b]$, since the bridge B would interlace with B' in such a case.

Part b): Let the span $[a, b]$ be non-degenerate and let the portion of P_i between a and b be $\langle a = a_1, \ldots, a_n = b\rangle$. Since there is no k-attachment bridge, $k > 2$, with span $[a,b]$, there must exist an a_i, $1 < i < k$ such that a, a_i, and b are in the same candidate list C, and no vertex outside $[a,b]$ is in C. Hence $\{a,b\}$ is an extremal separating pair. Also, since there is no bridge with attachments on a, b and some other vertex c outside $[a,b]$, there must be some vertex c on P_i such that either $c < a < b$ or $a < b < c$, and a, b, and c are in the same candidate list C'. Further, no vertex in the interval (a,b) can belong to C'. Hence $\{a,b\}$ is an adjacent pair in the candidate list C'.[]

Let us consider the case of a graph in which any pair of ear splits $e(a,b,i), e(c,d,j)$ with $i \neq j$ are disjoint. In this case we can perform the ear splits in Algorithm 3.2 corresponding to different ears in parallel. To process separating pairs on a single ear P_i we run the following algorithm.

Algorithm 3.3: Performing Ear Splits on a Single Ear

Input A biconnected graph G together with $D_i(P_i)$, the coalesced graph of the bridge graph of a nontrivial ear P_i in an open ear decomposition of G, with $P_i = \langle 0, 1, ..., k \rangle$.

Output: The split graphs of G after all Tutte splits on P_i have been performed.

vertex j, u, v, w, x, y; {∗ These vertices may be subscripted. ∗}

delete redundant two-attachment bridges;

pfor each attachment vertex v of each star B in $D_i \to$ make a copy v_B of v **rofp**;

pfor each internal vertex v on $P_i \to$

> **if** there is no star with an internal attachment on v $\to$ make an additional copy v_P of v to represent the lower split graph formed when all adjacent separating pairs containing v have been processed **fi**;

rofp;

pfor $j = 0$ **to** $k-1 \to$

> **if** there is no bridge with its leftmost attachment on j $\to$ replace edge $(j, j+1)$ on P_i by an edge incident on j_C, where C is B if there is a bridge B with an internal attachment on j and is P otherwise **fi**

rofp;

pfor $j = 1$ **to** $k \to$

> **if** there is no bridge with its rightmost attachment on j $\to$ replace edge $(j-1, j)$ on P_i by an edge incident on j_D, where D is B' if there is a bridge B' with an internal attachment on j and is P otherwise **fi**

rofp;

{∗ Process nonvacuous adjacent separating pairs. ∗}

pfor each star B in D_i $\rightarrow$

let the end attachments of B on P_i be v and w, $v < w$;

replace all edges in B incident on v by edges incident on v_B;

replace all edges in B incident on w by edges incident on w_B;

if B has no child-star B' with an attachment at v $\rightarrow$ replace edge $(v, v+1)$ on P by an edge incident on v_B **fi**;

if B has no child star B' with an attachment at w $\rightarrow$ replace edge $(w-1, w)$ by an edge incident on w_B **fi**;

place a virtual edge (v_B, w_B, i), and another virtual edge (v_C, w_D, i), where C (resp. D) is the parent-star of B if the parent star of B has an attachment at v (resp. w) and is P otherwise;

replace each internal attachment edge of B on a vertex u in P_i by an edge incident on u_B

rofp;

{∗ Process extremal pairs. ∗}

pfor each star B in D_i $\rightarrow$

let the attachments of B on P_i be $v_0 < v_1 < ... < v_l$;

pfor each j in $\{0, ..., l-1\}$ for which (v_{j_B}, v_{j+1_B}) is not an edge in the current component containing B $\rightarrow$

for convenience of notation let x denote v_j and let y denote v_{j+1};
make a copy x_{B_r} of x and a copy y_{B_l} of y;
replace the edge on P_i connecting x_B to the next larger vertex in the current graph by an edge incident on x_{B_r};
replace the edge on P_i connecting y_B to the next smaller vertex in the current graph by an edge incident on y_{B_l};
place a virtual edge (x_B, y_B, i) and another virtual edge (x_{B_r}, y_{B_l}, i)

rofp

rofp

end.

Algorithm 3.3 is an implementation of Algorithm 3.2 on ear P_i using the results of Lemmas 3.10, 3.11 and 3.12. We leave the proof of correctness of the algorithm to the reader. We also leave it to the reader to verify that all steps in the algorithm can be performed in logarithmic time with a linear number of processors in the size of D_i.

There are two problems with using this approach in an efficient logarithmic time algorithm for forming the triconnected components of a graph. One is that we are working with the D_i and the total size of these graphs need not be linear in the size of G. The second is that this approach will not work if a vertex a appears in an ear split for two different ears. For instance, two-attachment bridges corresponding to nonvacuous adjacent separating pairs will be separated on two different ears and this would cause processor conflicts. In Section 4.3 we show how to overcome these two problems to obtain logarithmic time parallel algorithm using a linear number of processors for finding the triconnected components of a general biconnected graph.

4 Efficient Implementation of Triconnectivity Algorithm

This section deals with a logarithmic time, linear processor implementation of Algorithms 3.1 and 3.2.

Section 4.1 gives such an algorithm for constructing the ear graphs of the nontrivial ears in an open ear decomposition (step 2 of Algorithm 3.1). Section 4.2 gives an algorithm with these bounds for constructing the coalesced graph of a star graph, and for extracting the candidate lists from its star embedding (steps 3 and 4 of Algorithm 3.1). In Section 4.3 we show that the results in sections 4.1 and 4.2 lead to a simple implementation of Algorithm 3.2 that runs in logarithmic time with a linear number of processors.

The algorithm in Section 4.1 for constructing the ear graphs is fairly intricate. A considerably simpler algorithm for this problem is given in Miller & Ramachandran [MR87] (exercise 4). However, although the algorithm in [MR87] is efficient, it needs $\log^2 n$ parallel time.

4.1 Forming the Ear Graphs

In this section we develop a parallel algorithm to find the ear graph of each nontrivial ear. This algorithm is based on material from Fussell, Ramachandran & Thurimella [FRT89], though the development here is somewhat different.

We begin by describing in Section 4.1.1 a simple linear processor, logarithmic time algorithm to find the bridge graph of each path in a collection of vertex-disjoint paths in a given graph. The set of nontrivial ears does not form a collection of vertex-disjoint paths since the endpoints of an ear are contained in other ears. Hence we cannot apply the algorithm in Section 4.1.1 to obtain the bridge graphs or ear graphs of nontrivial ears. However, in Sections 4.1.2 and 4.1.3 we present a collection of results that allow us to transform the input graph G, together with an open ear decomposition $D = [P_0, ..., P_{r-1}]$, into a modified graph G_l, together with a collection of edge-disjoint paths $[P'_0, ..., P'_{r-1}]$ with the useful property that the innard of each P'_i is P_i and the ear graph of each nontrivial ear P_i in D can be derived from the bridge graph of P_i in G_l. This property allows us to use the simple technique of section 4.1.1 on the innards of the P'_i, since these paths are vertex-disjoint.

The technique presented in section 4.1.3 is called the 'local replacement technique'.

4.1.1 Bridges of Disjoint Collection of Paths

In this section we present an algorithm for constructing the bridge graph of each path in a collection of vertex-disjoint paths in a graph.

Algorithm 4.1: Forming the Bridge Graph of Each Path in a Collection of Vertex-Disjoint Paths
Input: Graph $G = (V, E)$, together with a collection of vertex-disjoint paths $\{Q_0, ..., Q_{k-1}\}$.
Output: The bridge graph of each $Q_i, i = 0, ..., k-1$.

> **integer** i; **vertex** a, b, v; $\{*\ v$ will be subscripted by an integer. $*\}$
>
> **pfor** each $i \to$ collapse the vertices in Q_i into a vertex v_i **rofp**;
>
> let the resulting graph be G^-;
>
> **pfor** each $i \to$
>
>> **pfor** each block β of G^- with cutpoint $v_i \to$ form a nontrivial bridge B of Q_i with the edges of G^- in β that are incident on v_i as attachment edges **rofp**;
>>
>> **pfor** each edge (a, b) in $G-\ \{Q_i\}$ with a and b in $Q_i \to$ form a bridge of Q_i with attachments a and b **rofp**
>
> **rofp**

end.

It is straightforward to see that this algorithm correctly constructs the bridge graph of each of the Q_i, and that it runs in logarithmic time with a linear number of processors.

In the following sections we will use Algorithm 4.1 to find the ear graphs of the nontrivial ears in an open ear decomposition of a biconnected graph. We start by relating open ear decomposition to an st-graph in the next section.

4.1.2 The st-graph

Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$ with $P_0 = (s, t)$. Since G is biconnected, it has an *st* numbering (exercise 2).

Lemma 4.1 Let G be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$, where $P_0 = (s, t)$. Then it is possible to direct each ear in D from one endpoint to the other such that the resulting directed graph G_d is an *st* graph.

Proof We prove the lemma by establishing, by induction on i, that the graph $P_{0,i} = \cup_{j=0}^{i} P_j$ satisfies the statement of the lemma.

BASE: $i = 0$. Direct (s, t) from s to t.

INDUCTION STEP: Assume that the result is true until $i - 1$ and consider i.

Let D_{i-1} be the directed graph obtained from $P_{0,i-1}$ by directing its ears according to the statement of the lemma. Assume that the vertices in $P_{0,i-1}$ are numbered according to an *st* numbering consistent with D_{i-1}.

Let u and v be the endpoints of ear P_i and assume without loss of generality that u is numbered lower than v in the *st* numbering for $P_{0,i-1}$. Direct P_i from u to v.

We claim that $D_{i-1} \cup \{P_i$ directed from u to $v\}$ satisfies the statement of the lemma. This follows from the following construction. Number the internal vertices of P_i in order from u as $v, v+1, ..., v+k-1$, where k is the number of internal vertices of P_i. Replace the number of each vertex w in $P_{0,i-1}$ with $w \geq v$ by $w + k$. The resulting numbering is a valid *st* numbering for $P_{0,i}$ and $D_{i-1} \cup \{P_i$ directed from u to $v\}$ is its *st* graph.[]

Given an open ear decomposition $D = [P_0, ..., P_{r-1}]$, Maon, Schieber & Vishkin [MSV86] give a parallel algorithm to direct each ear in D as in Lemma 4.1 such that the resulting directed graph is an *st* graph. Let G_{st} be this graph, which we will call the *st-graph of D*. The graph T_{st}, the *st-tree of D,* is the directed spanning tree obtained from G_{st} by deleting the last edge in each ear except P_0. We can similarly construct G_{ts} and its directed spanning tree T_{ts} by considering P_0 to be directed from t to s. We will refer to G_{ts} as the *reverse directed graph of* G_{st} and vice versa.

We now state two simple but useful properties of open ear decomposition and the trees T_{st} and T_{ts}.

Property 4.1 Let P_i and P_j be two ears in an open ear decomposition D of graph G with $i < j$. Then, all vertices and edges of P_j belong to a single bridge of P_i in G.

Property 4.2 Let $p = \langle u_0, ..., u_i \rangle$ be a directed path in T_{st} or T_{ts}. Then the ear numbers of the vertices in p are nondecreasing when going from u_0 to u_i.

4.1.3 The Local Replacement Graph

In this section we describe a transformation of a biconnected graph G with an open ear decomposition $D = [P_0, ..., P_{r-1}]$ into a new graph G_l, called the *local replacement graph* of $(G; D)$. In the graph G_l, each ear P_i in G is converted into a path P'_i with the innard of P'_i being P_i and with the bridge graph of P_i in G_l corresponding to the ear graph of P_i in G.

Consider any vertex v in G. Let the degree of v be d ($d \geq 2$). Of the d edges incident on v, two belong to $P_{ear(v)}$. Each of the remaining $d-2$ edges incident on v is an end edge of some ear P_j, with $j > ear(v)$. In the local replacement graph G_l we will replace v by a rooted tree with $d-1$ vertices, with one vertex for each ear containing v. The root of this tree will be the copy of v for the ear containing v. The actual form of the tree is computed from T_{st} and T_{ts} as in the algorithm below. The tree representing vertex v will be called the *local tree of* v and will be denoted by T_v.

Algorithm 4.2: Constructing the Local Replacement Graph
Input:
A biconnected graph $G = (V, E)$;
an open ear decomposition $D = [P_0, ..., P_{r-1}]$ for G, with $P_0 = (s, t)$;
the st-graph G_{st} with its spanning tree T_{st} and the ts-graph G_{ts} with its spanning tree T_{ts}.
Output: The local replacement graph G_l of $(G; D)$.

integer i, j; {∗ These integers range in value from 0 to $r-1$. ∗}

vertex $a,\ q,\ u,\ v,\ w$; {∗ $q,\ u,\ v$ and w may be subscripted by an integer. ∗}

edge $a,\ e,\ f,\ n$; {∗ e and f will be subscripted by an integer. ∗}

rename each vertex v in G by v_j, where $ear(v) = j$;

{∗ We will refer to the vertex $v_{ear(v)}$ interchangeably as either v or $v_{ear(v)}$. ∗}

1. **pfor** each outgoing ear P_i at each vertex v in G_{st} →

    let the edge in P_i incident on v be e_i and let the nontree edge in P_i be f_i;

    detach edge e_i from v and label the detached endpoint as v_i;

    let a be a base edge of the fundamental cycle created by f_i in T_{st} with $ear(a) \neq i$;

    if $ear(a) \leq ear(v) \ \rightarrow v_{ear(v)} \ := \ parent(v_i)$

    $|$ $ear(a) > ear(v) \ \rightarrow \ v_{ear(a)} \ := \ parent(v_i)$ **fi** ;

    direct this edge from $parent(v_i)$ to v_i

rofp;

let the undirected version of the graph obtained in step 1 be G^1, the directed version be G^1_{st} and its associated spanning tree be T^1_{st} and the reverse directed graph be G^1_{ts} and its associated spanning tree be T^1_{ts};

2. repeat step 1 using G^1_{ts} and T^1_{ts} and let the resulting undirected graph be G^2, the resulting directed graph be G^2_{ts} and its associated spanning tree be T^2_{ts}, and the reverse directed graph be G^2_{st} and its associated spanning tree be T^2_{st};

   {* In the following we process parallel ears by constructing a new graph H. *}

   pfor each parallel ear $P_i \rightarrow$ **create** a vertex q_i **rofp** ;

   pfor each nontree edge n in $T^2_{st} \rightarrow$

   > **if** the base edges of the fundamental cycle of n belong to ears P_i and P_j, where P_i and P_j are parallel to each other $\rightarrow$ **create** an edge between q_i and q_j **fi**

   rofp;

   call the resulting graph H;

   find a spanning tree in each connected component of H and root it at the vertex corresponding to the minimum numbered ear in the connected component;

3. **pfor** each vertex q_i in H that is not a root of a spanning tree $\rightarrow$

   > let P_i be directed from endpoint u to endpoint w in G_{st}; let q_j be the parent of q_i in the spanning tree in H;
   >
   > replace the parent of u_i in T^2_{st} by u_j and the parent of w_i in T^2_{ts} by w_j

   rofp;

   denote the undirected version of the graph formed in step 3 by G_l, the directed graph from s to t by G'_{st} and its associated spanning tree by T'_{st} and the reverse directed graph by G'_{ts} and its associated spanning tree by T'_{ts}; call G_l the *local replacement graph* of G;

   call the underlying undirected tree constructed in steps 1, 2 and 3 from each vertex v in G the *local tree* T_v; call $v_{ear(v)}$ the root of T_v, and consider T_v to be an out-tree rooted at $v_{ear(v)}$. Call the part of T_v constructed by assigning parents in T^2_{st} the *o-tree* OT_v *of* T_v and the part of T_v constructed by assigning parents in T^2_{ts} the *i-tree* IT_v *of* T_v;

   {* In G^2_{st}, OT_v is an out-tree rooted at $v_{ear(v)}$ and IT_v is an in-tree rooted at $v_{ear(v)}$ and vice-versa in G^2_{ts}. *}

   denote by P'_i the ear P_i, together with the edge connecting each endpoint of P_i to its parent in its local tree in G_l;

   {* Note that the innard of P'_i (i.e., the path P'_i excluding its two end edges) is P_i. *}

   denote the first vertex on P'_i when directed as in G'_{st} by $L(P'_i)$, *the left endpoint of* P'_i, and the last vertex on P'_i when directed as in G'_{st} by $R(P'_i)$, *the right endpoint of* P'_i.

Figure 4.1: Constructing G_l from G

end.
An example of the construction in Algorithm 4.2 is shown in figure 4.1.

We will prove the following:

1. All ears with endpoints as descendant of v_i in T_v must belong to the same bridge of P_i in G.

2. An ear P_j with v_j not a descendant of v_i in T_v must be part of an anchor bridge of P_i or of a bridge of P_i with attachments to only the endpoints of P_i in G.

We start with the following preliminary lemmas.

Lemma 4.1 Let v_i be a proper ancestor of v_j in T_v, the local replacement tree of vertex v. Then either P_i and P_j are parallel to each other or $i < j$.
Proof Without loss of generality we assume that v_i and v_j belong to OT_v.

By the construction in Algorithm 4.2, either v_i is a proper ancestor of v_j in T^1_{st} or v_i and v_j are unrelated in T^1_{st} and v_i becomes a proper ancestor of v_j in step 3. In the latter case, v_i and v_j are parallel to each other and we are done. So for the rest of the proof we assume that v_i is a proper ancestor of v_j in T^1_{st}.

Let T_v^1 be the out-tree for vertex v at the end of step 1 of Algorithm 4.2. The vertex set of T_v^1 is $\{v_i \mid \text{vertex } v \text{ is contained in } P_i \text{ in } G\}$. We claim that the subscripts of the vertices are strictly increasing in any directed path in T_v^1. To see this, let v_k be the parent of v_j in T_v^1. If $k = ear(v)$ then $k < j$ since one endpoint of P_j in G is v. If $k \neq ear(v)$ let w be the other endpoint of P_j. By the construction in step 1 of Algorithm 4.2, w is a proper descendant of v in T_{st}. Hence by Property 4.2, $k \leq ear(w)$ and since $ear(w) < j$ we have $k < j$.

Hence if v_i is a proper ancestor of v_j in T_{st}^1 then $i < j$.[]

Definition Let (v, w) be the first edge on P_i in G_{st}. Then $T_{st}(i)$ is the subtree of T_{st} rooted at w. Similarly if (x, y) is the first edge on P_i in G_{ts} then $T_{ts}(i)$ is the subtree of T_{ts} rooted at y.

Lemma 4.2 Let v_i and v_j be vertices in T_v such that neither is a descendant of the other. Then in G, the following two properties hold.

a) Either P_i and P_j are ears parallel to each other, or $P_i \cap P_j = \{v\}$;

b) If $v_i \in OT_v$ then $P_j \cap T_{st}(i)$ =$\{v\}$ and if $v_i \in IT_v$ then $P_j \cap T_{ts}(i)$ =$\{v\}$.

Proof Exercise.[]

Lemma 4.3 Let v_i be a vertex in T_v and let
$S_i = \{\text{ears } P_j \text{ in } G \mid P_j \text{ contains } v \text{ and } v_j \text{ is not a proper descendant of } v_i \text{ in } T_v\}$.
Let v_k be a child of v_i in T_v and let T_k be the subtree of T_v rooted at v_k.
Then, all of the ears P_l in G such that v_l is in T_k belong to a single bridge of S_i in G.

Proof By induction on the height of T_k. We assume, without loss of generality that $v_i \in OT_v$.

BASE: Height of $T_k = 0$. Then T_k contains only one vertex and the claim is vacuously true since the corresponding ear P_k must belong to some single bridge of S_i (by Property 4.1 and Lemma 4.1 for those ears P_j in S_k with v_j an ancestor of v_i, and by Lemma 4.2, part a) for those P_j in S_k with v_j unrelated to v_i in T_v).

INDUCTION STEP: Assume that the lemma is true for height of T_k up to $h-1$ and let height of T_k be h. Let v_l be any child of v_k. Then T_l has height at most $h-1$ and hence by the induction hypothesis, all of the ears whose corresponding vertices lie in T_l belong to a single bridge of $S_i \cup \{P_k\}$ in G. Hence all of these ears belong to a single bridge B of S_i in G.

We now claim that bridge B contains ear P_k as well. The proof is a case analysis depending on whether v_k was made the parent of v_l in T_v in step 1 or in step 3 of Algorithm 4.2.

Case 1: v_k was made parent of v_l in step 1. Then P_k and P_l are not parallel to each other. Let (x, y) be the nontree edge (with respect to T_{st}) in ear P_l (figure 4.2a). Then by construction, y is a descendant of w, where (v, w) is the first edge on P_k in G (since $v_k \neq v_{ear(v)}$). But by Property 4.1, Lemma 4.1 and Lemma 4.2, none of the vertices on the tree path from w to y can be contained in an ear in S_i. Hence all vertices and edges in ear P_k belong to bridge B of S_i in G.

Figure 4.2: Illustrating the proof of Lemma 4.2

Case 2: v_k was made parent of v_l in step 3. Then P_k and P_l are parallel to each other. Further since v_k was made parent of v_l in step 3, there is a nontree edge n (with respect to T^2_{st}) whose fundamental cycle C contains both v_k and v_l (figure 4.2b). But none of the vertices in C other than the lca can belong to an ear in S_i by Lemma 4.2, part b, since all of these vertices are in either $T_{st}(k)$ or $T_{st}(l)$. Hence P_k is contained in bridge B of S_i in G.

This concludes the proof of the induction step and the lemma is proved.[]

Corollary to Lemma 4.3 Let v_i be a vertex in T_v and let v_j be a child of v_i in T_v. Then all ears P_k in G with v_k in T_j belong to a single bridge of P_i in G.

Proof This follows immediately from Lemma 4.3 by observing that P_i is contained in S_i.[]

Lemma 4.4 Let v_i be a vertex in T_v and let v_j be another vertex in T_v which is not a descendant of v_i. Then in G, P_j either belongs to the anchoring star of P_i or belongs to a bridge of P_i that has attachments only to the endpoints of P_i.

Proof Without loss of generality we assume that $v_i \in OT_v$. If $v_j \in IT_v$ then the outgoing edge e of P_j in G_{st} cannot be a descendant of v (since in that case G_{st} would contain a cycle). Further $P_i \cap P_j = \{v\}$ by Lemma 4.2. But then, there is a path from s to P_j in G that avoids ear P_i and hence P_j belongs to the anchoring star of P_i (since $ear(s) = 0$.)

For the rest of the proof we assume that $v_j \in OT_v$. Let $lca(v_i, v_j) = v_k$.

Case 1: $v_j = v_k$. If v_j is not parallel to v_i, then $i < j$ by Lemma 4.1, and hence P_j belongs to the anchoring star of P_i.

If P_j is parallel to P_i, let v_l be the root of the spanning tree of its connected

component in H formed in step 2 of Algorithm 4.2. By construction, v_l must be an ancestor of v_j.
Case 1.1: If $v_l = v_j$ then $j < i$ (since the spanning tree is rooted at the vertex with minimum index) and hence P_j is part of the anchoring star of P_i.
Case 1.2: If v_l is a proper ancestor of v_j then consider a sequence of nontree edges that caused the edges on the path from v_j to v_l in T_v to be placed in H. None of the vertices in the fundamental cycle of any of these nontree edges in G lie on P_i. Hence in G, these nontree edges, together with appropriate tree edges, induce a path from a vertex in P_j to a vertex in P_l that avoids all vertices in P_i. Hence P_j is in the same bridge of P_i as P_l and hence belongs to the anchoring star of P_i (since l must be less than i).
Case 2: $v_j \neq v_k$.

In this case v_k is a proper ancestor of v_j. Let v_m be the child of v_k that is an ancestor of v_j. Then all ears with corresponding vertices in T_m lie on a single bridge of P_i (by Corollary to Lemma 4.3).

Let v_l be the nearest ancestor of v_i such that P_l is not parallel to P_i.
Case 2.1: v_l is a proper descendant of v_k.

In this case, P_m is not parallel to P_i, since otherwise, by step 3 of Algorithm 4.2, v_m would be a descendant of v_l. Also, by Lemma 4.2, $P_m \cap P_i = \{v\}$. Finally, the nontree edge in P_m completes a fundamental cycle in G, one of whose base edges belongs to some $P_q, q \leq k$. None of the vertices other than v in this fundamental cycle belongs to P_i, since by step 1 of Algorithm 4.2, the two base edges in the fundamental cycle of which P_i is part, belong to P_i and P_l. Hence, P_m (and thus P_j) belongs to the same bridge of P_i as P_q and is thus part of the anchoring star of P_i.
Case 2.2: $v_l = v_k$ (the nontrivial case).

Let y be the last vertex on P_i and let z be the child of v_k in T^2_{st} that is an ancestor of y. By construction (step 1 of Algorithm 4.2), either $v_k = v_{ear(v)}$ or z lies in $T_{st}(k)$.
Case 2.2.1: If v_m is not parallel to v_i then let (w, x) be the last edge in P_m in G. The vertex x is contained in P_q, for some $q \leq k$ and x is not contained in P_i (by Lemma 4.2, part a). If x lies on the path from v to y then P_m (and hence P_j) is part of the same bridge of P_i as P_k and hence is part of the anchoring star of P_i. Otherwise, x is not an ancestor of y and by the st-numbering property, there is a path from x to t (and hence to s) in G that avoids all vertices in P_i. Hence again we have the case that P_j is part of the anchoring star of P_i.

Now consider the case when P_m is parallel to P_i and assume that P_m (and hence P_j) is part of a bridge B of P_i with an internal attachment on P_i. We will show that B must be an anchor bridge of P_i.

Since B has an internal attachment on P_i, there is a path p in G_l from v_m to some vertex u that is internal to P_i that avoids all other vertices in P_i. The path p must contain at least one nontree edge whose lca is $\leq v_k$. Let n be the first such nontree edge encountered when traversing p from v_m to u.
Case 2.2.2: $lca(n) < v_k$ in T'_{st}. Then there is a path in G from a vertex in P_m to

$lca(n)$ that avoids P_i and since $k < i$, $lca(n)$ belongs to an ear numbered less than i (by Property 4.1). Hence B is an anchor bridge of P_i.
Case 2.2.3: $lca(n) = v_k$ in T'_{st}. Let $e = (v_k, v_m)$ and $f = (v_k, v_n)$ be the base edges of the fundamental cycle of n in T^2_{st}. Then P_n cannot be parallel to P_m, since otherwise v_m and v_n would be in the same connected component of H and hence would be in a single subtree rooted at a child of v_k. But if P_n is not parallel to P_m, it is also not parallel to P_i, and we can use the analysis used with P_m in case 2.2.1 to deduce that P_n is part of an anchor bridge of P_i. Hence B is an anchor bridge of P_i.
Case 2.3: v_l is a proper ancestor of v_k.

The analysis of this case is similar to case 1.2 and we can deduce that P_j is part of the anchoring star of P_i.

This concludes the case analysis and the lemma is proved.[]

Theorem 4.1 Let G be a biconnected graph with an open ear decomposition $D = [P_0, ..., P_{r-1}]$ and let P_i be a nontrivial ear in D. Let B be a bridge of P_i in G_l and let a and b be any two edges of G that are in B. Then,

a) If B contains the endpoints of P'_i in G_l then a and b belong to the anchoring star of P_i in G.

b) If B does not contain the endpoints of P'_i in G_l then a and b are both part of a single nonanchor bridge of P_i in G.

Also, if c and d are two edges of G that do not belong to a single bridge of P_i in G_l then c and d belong to different bridges of P_i in G.

Proof The theorem follows from observing that any additional connectivity induced in G_l - $\{P_i\}$ that is not present in $G-$ $\{P_i\}$ must occur at T_v, for some vertices $v \in P_i$, and by applying the Corollary to Lemma 4.3 and Lemma 4.4.[]

Corollary to Theorem 4.1 Let G'_i be the bridge graph of P_i in G_l. Let G_i be obtained from G'_i by replacing all multiple edges in G'_i by a single copy. Then G_i is the ear graph of P_i.

By the above results, the following algorithm constructs the ear graph of each nontrivial ear in an open ear decomposition $D = [P_0, ..., P_{r-1}]$ of a biconnected graph G.

Algorithm 4.3: Constructing the Ear Graphs
Input: A biconnected graph $G = (V, E)$ together with an open ear decomposition $D = [P_0, P_1, ..., P_{r-1}]$.
Output: The ear graph of each nontrivial ear in D.

form the local replacement graph G_l of G, together with the associated paths $P'_0, ..., P'_{r-1}$ using Algorithm 4.2;

apply Algorithm 4.1 to G_l with the nontrivial ears in D as the vertex-disjoint paths to obtain the bridge graph of each nontrivial P_i in G_l;

pfor each nontrivial ear $P_i \rightarrow$ obtain the ear graph G_i of P_i in G from the bridge graph of P_i in G_l by replacing all multiple edges by a single copy **rofp**

end.

We leave it as an exercise to verify that all steps in Algorithm 4.3 can be performed in logarithmic time with a linear number of processors.

4.2 Finding the Candidate Lists

In this section we describe an efficient algorithm to implement steps 3 and 4 in Algorithm 3.1. Given a star graph $G(P)$, steps 3 and 4 require us to find its coalesced graph $G_c(P)$ and extract the candidate lists from its star embedding.

We present the algorithm for forming the coalesced graph, which is based on material in Ramachandran & Vishkin [RV88], in two parts. In Section 4.2.1 we present an algorithm to find the coalesced graph when every star in $G(P)$ has exactly two attachments. Then in Section 4.2.2 we give an efficient reduction from a general star graph to this special case. The reduction presented here is somewhat different from the one in [RV88].

Our algorithm solves a more general problem than that of finding the coalesced graph of a star graph $G(P)$: It also provides an 'interlacing parity' (which is defined in Section 4.2.1) for every pair of stars on P. This property is useful in determining planarity of $G(P)$ for the case when every star has to be embedded completely on one side of P. While this is not needed for the triconnectivity algorithm, it is an important step in the parallel algorithm for graph planarity given in Ramachandran & Reif [RR89].

In Section 4.2.3 we describe a simple efficient algorithm to form the star embedding of $G_c(P)$ and to extract from it the candidate lists of $G(P)$.

4.2.1 Determining Interlacings of Chords on a Path

Let $G(P)$ be a star graph in which each star has exactly two attachments on P. For simplicity we assume $P = \langle 0, ..., n \rangle$. Then G can be viewed as the simple path P together with a collection of chords (i, j) on P. We shall refer to such a graph as a *chord graph.*

Let $a = (u, v)$ and $b = (w, x)$ be two chords on P, where $u < v$ and $w < x$. If a and b interlace, then they cannot be placed on the same side of P in a planar embedding. If a and b do not interlace, then they can be placed in a planar embedding on the same (opposite) side of C if and only if there exists no sequence of chords $\langle a = a_0, a_1, ..., a_r = b \rangle$, with r odd (even) such that a_i interlaces with a_{i+1}, $0 \leq i \leq r - 1$. If there is such a sequence with r even then a and b have *even interlacing parity* and if there is such a sequence with r odd, then a and b have *odd interlacing parity.* If no such sequence exists for r either odd or even, then a and b have *null interlacing parity:* in this case a and b can be placed either in the same side or in opposite sides of P in a planar embedding. It is possible for a and b to have both odd and even parity – in this case, no planar embedding of G is possible if every chord is to be placed completely on one side of P.

We now present an efficient parallel algorithm for preprocessing the graph G so that an interlacing parity for any pair of chords can be determined in constant time. If a pair of chords a and b have both even and odd interlacing parities, then we will find only one of these parities. A high level description of the algorithm is as follows: We construct an auxiliary graph G_I called the *interlacing parity graph of* G with a vertex for each chord on P. We then place some edges in G_I. Each such edge connects a pair of vertices whose corresponding chords interlace. We do not put in an edge for every pair of interlacing chords, but only a subset of them, so that the size of G_I is linear in the size of G. In particular, if r is the number of chords on P then G_I will have at most $2r$ edges. Further, if a and b are two chords for which there exists some interlacing sequence then a and b will lie in the same connected component of G_I. Since each edge in G_I represents an actual interlacing, we can obtain an interlacing parity for each pair of chords with an interlacing sequence by finding a spanning tree in each connected component of G_I and two coloring the vertices of the spanning tree: Now, two chords, whose vertices are in the same connected component in G_I, have odd interlacing parity if they have different colors and even parity if they have the same color. Note that to form the coalesced graph of $G(P)$, we only need to find the connected components of G_I, and coalesce all chords that correspond to vertices in each connected component.

The algorithm is presented below. Since vertices in G_I correspond to chords on P, we will sometimes refer to a vertex in G_I as a chord; by this we mean the chord in G that this vertex represents.

Algorithm 4.4: Interlacing Parity for a Chord Graph
Input: Undirected graph G consisting of a simple path $P = \langle 0, ..., n \rangle$, together with a collection of chords on P.
Output: A label on each chord which allows an interlacing parity of any pair of chords to be determined in constant time by one processor.

vertex $u,\ v$; **edge** $c,\ l,\ r$; $\{*$ $u,\ v,\ l,\ r$ may be subscripted. $*\}$

1. **pfor** each chord $c = (u, v)$, $u < v$, that interlaces with some other chord $\rightarrow$

    $\{*$ Left rule. $*\}$

    let u_l be the minimum numbered vertex on P such that c interlaces with a chord incident on u_l;

    if $u_l < u$ $\rightarrow$ find the chord $l_c = (u_l, v_l)$ with maximum v_l that interlaces with c and place an edge (the *left edge*) in G_I between vertices c and l_c **fi**;

    $\{*$ Right rule. $*\}$

let v_r be the maximum numbered vertex on P such that c interlaces with a chord incident on v_r;

if $v_r > v \ \rightarrow$ find the chord $r_c = (u_r, v_r)$ with minimum u_r that interlaces with c and place an edge (the *right edge*) in G_I between c and r_c **fi**

rofp;

find a spanning tree in each connected component of the interlacing parity graph G_I and two-color the spanning trees;

assign a label $\langle$component number, color$\rangle$ to each vertex;

pfor each chord in $G(P) \rightarrow$ assign the label of the vertex in G_I corresponding to it **rofp**

end.

With this preprocessing we can determine an interlacing parity for any pair of chords c, d on P as follows: If component number of c is not equal to component number of d then c and d have null interlacing parity, otherwise they have even interlacing parity if they have the same color, and odd interlacing parity if they have different colors.

Lemma 4.5 . If a pair of chords α, β have an interlacing parity that is not null, then α and β appear in the same connected component of G_I.

Proof. By induction on the number of chords r on P.

BASE: $r = 2$. This is immediate.

INDUCTION STEP: Assume that the claim is true for all simple paths with up to $r-1$ chords, and let G be a graph consisting of a simple path P, together with r chords. Let v be the lowest numbered vertex on P that has a chord incident on it, and let $a = (v, w)$ be the chord incident on v with maximum w. Delete a from G to form G'. By the induction hypothesis, every pair of chords that have an interlacing parity that is not null in G' appear in the same connected component in G'_I.

We will show two things:

A. Any pair of chords α, β that lie in the same connected component in G'_I continue to lie in the same connected component in G_I.

B. Any chord having an interlacing parity with a will be in the connected component of a in G_I.

Lemma 4.5 follows. To see this consider any pair of chords α, β, where $\alpha \neq a$ and $\beta \neq a$. Assume α and β have an interlacing parity in G and consider an interlacing sequence. If the interlacing sequence does not include a apply claim A. If the interlacing includes a, then we may assume that a appears only once. Apply claim B to show that a is in the same connected component as its predecessor and successor in the sequence. Finally, apply claim A.

We show A. For this, observe that the only edges in G'_I that are not present in G_I are the edges introduced by the left rule for chords that interlace with a: each such edge is replaced by an edge connecting the chord to vertex a. Let b be such a chord interlacing with a, let its left edge in G'_I be (b, c). Its left edge in G_I is (b, a). We claim that a, b and c lie in the same connected component in G_I.
Case 1: Chord c interlaces with chord a. Then edges (a, c) and (a, b) are present in G_1 and hence a, b and c belong to the same connected component in G_1.
Case 2: Chord c does not interlace with a. Consider the right edge (c, d) of c. Then chord d has its right endpoint at least as large as the right endpoint of b, and hence d interlaces with a. Hence edges $(a, d), (c, d)$ and (b, a) are present in G_I, i.e., a, b and c lie in the same connected component in G_I.

We show claim B. For each chord b having an interlacing parity with a, consider an interlacing sequence $a_0 = a, a_1, ..., a_k = b$. Chords a_1 and b are in the same connected component of G_I by claim A. It suffices to show that chords a and a_1, which actually interlace, are in the same connected component. But, there must be an edge connecting them in G_I by the left rule for a_1. Claim B and Lemma 4.5 follow. []

Lemma 4.6 Algorithm 4.4 correctly finds an interlacing parity for each pair of chords on P.
Proof Since an edge (a, b) is placed in G_I only if chords a and b interlace, it follows that the algorithm finds a correct interlacing parity for every pair of chords that belong to the same connected component. By Lemma 4.5, vertices that belong to different connected components correspond to chords that have null interlacing parity. This establishes the correctness of the algorithm.[]

To implement step 1 we determine at each vertex v, the chord with smallest attachment s_v and the chord with largest attachment l_v incident on v, and we store at v the ordered pairs $(s_v, -v)$ and (l_v, v). With this preprocessing it is a simple exercise to verify that all steps of Algorithm 4.4 can be performed in logarithmic time with a linear number of processors on a CRCW PRAM.

4.2.2 Determining Interlacings of Stars in a Star Graph

In this section we consider a general star graph $G(P)$. We replace each star in $G(P)$ with a collection of chords and construct the interlacing parity graph for this new graph as in the previous section. We then add in some additional vertices and edges to this graph and we establish that the resulting graph can be used to obtain the interlacing parity of each pair of stars in $G(P)$ efficiently.

We now describe our construction. Let $P = \langle 0, 1, ..., n \rangle$. We replace each star S on $G(P)$ by a collection of chords as follows: Let the attachments of S on P be $a_0, a_1, ..., a_k$ with $a_0 < a_1 < ... < a_k$. We replace S by the chords $(a_0, a_i), i = 1, ..., k$ and the chords $(a_i, a_k), i = 1, ..., k-1$. We will refer to these chords as the *chords of* S.

Let $H(P)$ be the graph obtained from $G(P)$ by replacing each star in $G(P)$ by a collection of chords as described above. Let $H_I = (V, E)$ be the interlacing parity graph of $H(P)$. We construct $G_I = (V', E')$, the *interlacing parity graph of* $G(P)$ as follows:
$V' = V \cup \{v_S \mid S \text{ is a star in } G(P)\}$;
$E' = E_1 \cup \{(v_S, u) \mid S \text{ is a star in } G(P) \text{ and } u \in V \text{ represents a chord of } S\} \cup F$,
where E_1 and F is defined as follows:
$E_1 \; = \; E - \{(u, v) \in E \mid u \text{ and } v \text{ are chords of the same star } S \text{ in } G(P)\}$;
For each vertex i on P let
$F_i = \{(v_S, v_T) \mid S$ is a star in $G(P)$ with an internal attachment on i and T ranges over all other stars in $G(P)$ with an internal attachment on $i\}$
Then $F = \cup_{i=1}^{n-1} F_i$.
Lemma 4.7 Let S and T be two stars that interlace in $G(P)$. Then either S and T share an internal attachment on $G(P)$ or there exist a chord of S and a chord of T that interlace on P.
Proof Let S and T interlace on P. If they interlace by virtue of three common attachments then consider the middle attachment of these three. This attachment must be an internal attachment of both S and T.

If S and T interlace because there exist four vertices $a < b < c < d$ on P such that a and c are attachments of S and b and d are attachments of T, then the four vertices $a' < b < c < d'$, where a' is the first attachment of S and d' is the last attachment of T, also represent the interlacement of S and T. But (a', c) is a chord of S and (b, d') is a chord of T and (a', c) and (b, d') interlace.

A similar argument applies to the case when a and c are attachments of T and b and d are attachments of S.[]
Lemma 4.8 Let S and T be two stars that do not interlace on $G(P)$. Then S and T share no internal attachment on P, and for any pair of chords c and d, with c a chord of S and d a chord of T, c and d do not interlace on P.
Proof Exercise 8.[]
Lemma 4.9 Let G_I be the interlacing parity graph of a star graph $G(P)$. The following properties hold:

a. All vertices representing chords of a single star in $G(P)$ belong to a single connected component in $G(P)$.

b. Two stars S and T in $G(P)$ have null interlacing parity if and only if the vertices corresponding to their chords lie in different connected components in G_I.

Proof Exercise 8.[]
Lemma 4.10 Let $G(P)$ be a star graph and let S and T be two stars on $G(P)$. Let c be a chord of S and d a chord of T and let their corresponding vertices lie in a single connected component C of G_I. Let $\mathbf{X}$ be a two-coloring of a spanning tree of C. Then if the vertices corresponding to c and d have the same color in $\mathbf{X}$ then stars

S and T have even parity in $G(P)$ and if they have different colors then S and T have odd parity in $G(P)$.
Proof Exercise 8.[]

We now present the algorithm to determine interlacing parity for a general star graph.

Algorithm 4.5: Interlacing Parity for a Star Graph
Input: A star graph $G(P)$.
Output: A label on each star which allows an interlacing parity of any pair of stars to be determined in constant time by one processor.

> construct the interlacing parity graph G_I of G as described above;
>
> find a spanning tree in each connected component of G_I and two-color the spanning trees;
>
> assign a label $\langle$component number, color$\rangle$ to each vertex in G_I;
>
> **pfor** each star S in $G(P) \to$ assign the label of a vertex in G_I corresponding one of its chords **rofp**

end.

To determine an interlacing parity for any pair of stars S, T in $G(P)$ we proceed as in the previous section: If component number of S is not equal to component number of T then S and T have null interlacing parity, otherwise they have even interlacing parity if they have the same color, and odd interlacing parity if they have different colors.

Theorem 4.2 Algorithm 4.5 correctly determines the interlacing parity of any pair of stars in $G(P)$.
Proof The proof is a straightforward consequence of Lemmas 4.9 and 4.10.[]

Finally, to form the coalesced graph $G_c(P)$ of $G(P)$, we replace all stars in each connected component of G_I by a new star whose attachments are the union of the attachments of these stars.

4.2.3 The Star Embedding and the Candidate Lists

In this section we give a method to find the candidate lists in a star graph $G(P)$, given its coalesced graph $G_c(P)$. The algorithm forms the star embedding of $G_c(P)$ and extracts the candidate lists as those sets of vertices on P that lie on a single face in the star embedding of $G_c(P)$.

We first give a combinatorial characterization of a graph embedding that was introduced by Edmonds [Ed60]. Let $G = (V, E)$ be the graph to be embedded, and let D_G be the directed graph obtained from G by replacing each undirected edge (u, v) in E by two directed edges (u, v) and (v, u). A *combinatorial embedding, $I(G)$,* of the graph G is an assignment of a cyclic ordering to the set of outgoing edges from each vertex in the graph D_G. For each edge (u, v) in D_G let $next(u, v)$ be the edge (v, w) that follows edge (v, u) in the cyclic ordering of edges outgoing from vertex v. Then the graph defined by the next pointers is a collection of edge-disjoint cycles in D_G whose union is the edge set of D_G. Each cycle defined by the next pointers represents a *face* of $I(G)$. Let $n = |V|, m = |E|$, let c be the number of connected components in G and let f be the number of faces in $I(G)$. If Euler's formula $n - m + f = 1 + c$ is satisfied, then the combinatorial embedding $I(G)$ defines a planar embedding of G with the edges incident on each vertex embedded according to the cyclic ordering, and with each face in $I(G)$ representing a face in the planar embedding of G.

Our algorithm will find such a combinatorial embedding for $G_c(P)$ with the additional property that all stars in $G_c(P)$ lie on the same side of $G_c(P)$. This will give us a star embedding of $G_c(P)$. From this star embedding we can obtain the faces of $G_c^*(P)$ by following the next pointers as in the above definition, and we can obtain the candidate lists as those sequences of vertices of P that lie on a single face in the star embedding. The algorithm is given below.

Algorithm 4.6: Finding Candidate Lists
Input: The coalesced graph $G_c(P)$ of a star graph $G(P)$, with $P = \langle 0, 1, ..., n \rangle$.
Output: The candidate lists of $G(P)$.

vertex $c,\ u,\ v$; $\{*$ c will be subscripted. $*\}$

edge e;

interpret each edge (u, v) in $G_c(P)$ as a pair of directed edges (u, v) and (v, u);

1. **pfor** each vertex i on $P \rightarrow$

   sort the attachment edges on vertex i that represent the last attachment edge of their star in nonincreasing order of the first attachment of the star containing the edge; break ties by placing an attachment edge of a two-attachment star *after* an attachment of a star with three or more attachments;
   let the resulting array be A_i;
   $B_i\ :=\ \phi$;

   if there is a star with an internal attachment edge e on $i\ \rightarrow\ B_i\ :=\ e$ **fi**;
   sort the attachment edges on i that represent the first attachment of their star in nonincreasing order of the last attachment edge of the star containing the edge; break ties by placing an attachment on i of a two-attachment star *before* an attachment of a star with three or more attachments;
   let the resulting array be C_i;
   rearrange the adjacency list for i by concatenating edge $(i, i-1)$ (if it exists), arrays A_i, B_i, C_i and edge $(i, i+1)$ (if it exists) in this order;

make this list cyclical by causing the first edge on the list to follow the last edge

rofp;

2. **pfor** each star $S \rightarrow$

    rearrange the adjacency list for its center c_S in nondecreasing order of the attachment vertices;

    make this list cyclical by causing the first edge on the list to follow the last edge

    rofp ;

    form the star embedding of $G_c(P)$ by embedding P and by embedding the edges incident on vertex $i, i = 0, ..., n$ and on the center of each star in $G(P)$ cyclically according to their order in the rearranged adjacency lists obtained in steps 1 and 2;

3. **pfor** each edge (u, v) in $G_c(P) \rightarrow$ $next(u, v)$:= the edge following (v, u) in the adjacency list of v **rofp**;

4. use pointer jumping on the next pointers to partition the directed edges into cycles;

    pfor each cycle formed in step 1 $\rightarrow$ find the list of vertices on P that are contained in the cycle and output this sequence as a candidate list **rofp**

end.

Theorem 4.3 Algorithm 4.6 correctly finds the candidate lists of $G(P)$.

Proof We leave it as an exercise to verify that the combinatorial embedding in step 3 represents the star embedding of $G_c(P)$. The proof of correctness then follows from the properties of a combinatorial embeddings, together with the results in Section 3.3.[]

4.3 Finding Triconnected Components Efficiently

In this section we give an efficient parallel algorithm to find the triconnected components of a biconnected graph using Algorithm 3.3. Algorithm 3.3, when used in parallel on all nontrivial ears, finds the triconnected components of a biconnected graph G with an open ear decomposition D when no vertex is part of pairs separating two different ears. This property need not hold for a general biconnected graph G but if we use G_l, the local replacement graph of $(G; D)$, all separating pairs in G_l corresponding to separating pairs of G are internal to their corresponding paths P_i' and hence this property holds. Further, the size of all of the bridge graphs of the nontrivial paths P_i in G_l is linear in the size of G since these paths are vertex-disjoint in G_l. Hence we can find the triconnected components of G using the following simple algorithm:

Algorithm 4.7: Efficient Triconnected Components Algorithm
Input: Biconnected graph $G = (V, E)$ together with an open ear decomposition $D = [P_0, ..., P_{r-1}]$ for G.
Output: The triconnected components of G.

> form G_l from G using Algorithm 4.2;
>
> form the bridge graph $D_i'(P_i)$ in G_l of each nontrivial path P_i in D using Algorithm 4.1;
>
> **pfor** each nontrivial ear $P_i \rightarrow$ apply Algorithm 3.3 to D_i' to perform all of the ear splits on P_i **rofp**;
>
> **pfor** each connected component in the resulting graph $\rightarrow$ collapse all vertices in each local tree to get back a triconnected component of G **rofp**

end.

5 Towards Optimality

Although we have stated the parallel bound for the various steps in our algorithms as $O(\log n)$ parallel time with a linear number of processors, most of the steps can, in fact, be performed optimally in $O(\log n)$ time. Only the need to find connected components in a graph and to perform bucket sort prevents us from obtaining true optimality. In this context we note the following.

1) *Finding connected components*: We need to find connected components at several places in our algorithms. At present there is no optimal $O(\log n)$ time parallel algorithm known for graph connectivity although the algorithm in [CV86] is 'almost optimal'. However, this algorithm assumes that the graph is represented by its adjacency lists. Even if we assume that the input graph is represented by its adjacency lists, we still need to ensure that the adjacency lists for the various derived graphs used in the algorithms can be obtained optimally. For the open ear decomposition algorithm, the details of this construction are worked out in [Sc87]. The adjacency lists for some of the derived graphs used in the triconnectivity algorithm can also be obtained optimally. In the absence of an optimal algorithm for finding the adjacency list for a derived graph, we can use bucket sort in the range $[1..n]$, where n is the number of vertices in the graph, to rearrange an unordered edge list into adjacency lists for the individual vertices. This can be done in $O(\log n)$ time using $O((n+m) \cdot \log\log n / \log n)$ processors using the algorithm of Hagerup [Ha], where m is the number of edges in the graph. This is also the best bound known for testing graph connectivity if the input graph is specified as a list of edges that are not organized as adjacency lists for vertices.

The connected components of a graph can be obtained optimally in logarithmic time on a CRCW PRAM using the *randomized* algorithm of Gazit [Ga86]. The required adjacency lists can also be obtained optimally in logarithmic time using a randomized algorithm for bucket sort in the range $[1..n]$ given in [RR89b].

2) *Sorting* Our algorithms use sorting in several places. All of the sorting that is needed can be performed using an algorithm for bucket sort in the range $[1..n^2]$ and this can be done using the algorithm of [Ha] with the performance bound stated in 1). Further, in some cases the sorting step can be replaced by a more sophisticated algorithm that runs optimally (see, e.g., [FRT89]).

   In the ear decomposition and open ear decomposition algorithms, sorting is needed only if we require consecutive ear numbers. In most applications (including triconnectivity, four-connectivity and planarity), any sequence of increasing labels from a totally ordered set suffices for the ear labels, and in such cases, no sorting is needed in either of these algorithms.

6 Conclusion

In this report we have presented efficient parallel algorithms for testing graph biconnectivity and triconnectivity and for finding the triconnected components of a biconnected graph. All of these algorithms run in linear sequential time and thus represent new linear time sequential algorithms for these problems.

The algorithms given here differ significantly in structure from the earlier linear time algorithms for these problems which were based on depth-first search. For instance, our algorithms are very modular. Thus an implementer can choose a trade-off between efficiency of implementation and ease of programming in deciding which of several methods to use to implement each step in the various algorithms. This is in contrast to the earlier algorithms in which most of the steps were tied to a depth-first search of the input graph.

The algorithms given here are somewhat more complex than the earlier sequential algorithms. Parallel algorithm design is a challenging task, and in asking for an efficient parallel logarithmic time algorithm whose sequential implementation runs in linear time, we are requiring much more of the algorithm designer than we do of a designer of linear sequential algorithms. Hence it is not surprising that the parallel algorithms given here are rather complex.

Much of the additional complexity in the parallel algorithms of this report relative to the sequential ones is due to the fact that the parallel algorithms is unable to generate a depth-first search tree efficiently. We demonstrated this in the case of finding an open ear decomposition. However, due to the difference in the techniques used in the design of sequential and parallel algorithms, we sometimes obtain a simplified sequential algorithm from a parallel algorithm for a problem. For instance, in the case of finding triconnected components, the algorithm in this report is actually

simpler than the earlier linear time sequential algorithm in that it directly performs Tutte splits and hence does not need to perform any recombinations.
Acknowledgement I would like to thank the Austin Tuesday Afternoon Club for several comments on an earlier version of Section 2. I am especially grateful to Edsger W. Dijkstra and Jay Misra for their comments on algorithmic notation and specification of algorithms.

Exercises

1. Prove that a graph has an open ear decomposition if and only if it is biconnected.

2. Prove that a graph is biconnected if and only if it has an st-numbering.

3. Prove that the coalesced graph of a star graph is unique.

4. Let G be a biconnected graph with an open ear decomposition D and let G have n vertices and m edges. Use a divide and conquer approach to find the ear graph of each nontrivial ear in D in $O(\log^2 n)$ time with $O(n+m)$ processors.

5. Adapt the theorems and results in Section 3.3 to the problem of determining three-edge connectivity in an undirected graph. Simplify the results wherever possible.

6. Complete the proofs of lemmas in Section 4.1 by considering the case when vertices are in their corresponding in-trees.

7. Explain why it suffices to consider only T_{st} in step 3 of Algorithm 4.2.

8. Prove the correctness of Algorithm 4.5 by supplying the proofs of Lemmas 4.8, 4.9 and 4.10.

9. The *tree of triconnected components* of a biconnected graph G is a tree $T = (V, E)$ whose vertex set is the set of triconnected components of G, and which contains an edge $(x, y) \in E$ whenever the triconnected components x and y have the same copy of an edge introduced during a Tutte split.

   Give a parallel algorithm that runs in logarithmic time with a linear number of processors to find the tree of triconnected components of a biconnected graph.

REFERENCES

[Co88] R. Cole, "Parallel merge sort," *SIAM J. Comput.,* vol. 17, 1988, pp. 770-785.

[CV86] R. Cole, U. Vishkin, "Approximate and exact parallel techniques with applications to list, tree and graph problems," *Proc. 27th Ann. IEEE Symp. on Foundations of Comp. Sci.,* 1986, pp. 478-491.

[Ed60] J. Edmonds, "A combinatorial representation for polyhedral surfaces," *Not. Am. Math. Soc.,* vol. 7, 1960, p. 646.

[Ev79] S. Even, *Graph Algorithms,* Computer Science Press, Rockville, MD, 1979.

[FRT89] D. Fussell, V. Ramachandran, R. Thurimella, "Finding triconnected components by local replacements," *Proc. ICALP 89,* Springer Verlag LNCS 372, 1989, pp. 379-393; *SIAM J. Comput*, to appear.

[Ga86] H. Gazit, "An optimal randomized parallel algorithm for finding connected components in a graph," *Proc. 27th Ann. IEEE Symp. on Foundations of Comp. Sci.,* 1986, pp. 492-501.

[Ha87] T. Hagerup, "Towards optimal parallel bucket sorting," *Inform. and Comput.,* vol. 75, 1987, pp. 39-51.

[HT72] J. E. Hopcroft, R. E. Tarjan, "Finding the triconnected components of a graph," TR 72-140, Computer Science Department, Cornell University, Ithaca, NY, 1972.

[HT73] J. E. Hopcroft, R. E. Tarjan, "Dividing a graph into triconnected components," *SIAM J. Comput.,* 1973, pp. 135-158.

[KR91] A. Kanevsky, V. Ramachandran, "Improved algorithms for graph four-connectivity," *Jour. Comput. Syst. Sci.*, vol. 42, 1991, pp. 288-306.

[KR90] R. M . Karp, V. Ramachandran, "Parallel algorithms for shared memory machines," *Handbook of Theoretical Computer Science,* J. Van Leeuwen, ed., North Holland, 1990, pp. 869-941.

[KD88] S. R. Kosaraju, A. L. Delcher, "Optimal parallel evaluation of tree-structured computations by raking," *Proc. 3rd Aegean Workshop on Computing,* Springer-Verlag LNCS 319, 1988, pp. 101-110.

[Lo85] L. Lòvasz,"Computing ears and branchings in parallel," *Proc. 26th IEEE Ann. Symp. on Foundations of Comp. Sci.,* 1985, pp. 464-467.

[MSV86] Y. Maon, B. Schieber, U. Vishkin, "Parallel ear decomposition search (EDS) and st-numbering in graphs," *Theoretical Comput. Sci.,* vol. 47, 1986, pp. 277-298.

[MR86] G. L. Miller, V. Ramachandran, "Efficient parallel ear decomposition with applications," manuscript, MSRI, Berkeley, CA, January 1986.

[MR87] G. L. Miller, V. Ramachandran, "A new graph triconnectivity algorithm and its parallelization," *Proc. 19th Annual ACM Symp. on Theory of Computing,* 1987, pp. 254-263, *Combinatorica*, to appear.

[RR89] V. Ramachandran, J. H. Reif, "Planarity testing in parallel," TR-90-15, Dept. of Computer Sciences, The University of Texas, Austin, TX, 1990; preliminary version appears as "An optimal parallel algorithm for graph planarity," *Proc. 30th Ann. IEEE Symp. on Foundations of Comp. Sci.,* 1989, pp. 282-287.

[RR89b] S. Rajasekharan, J. H. Reif, "Optimal and sublogarithmic time randomized parallel sorting algorithms," *SIAM J. Comput.,* vol. 18, 1989, pp. 594-607.

[RV88] V. Ramachandran, U. Vishkin, "Efficient parallel triconnectivity in logarithmic time," *VLSI Algorithms and Architectures,* Springer Verlag LNCS 319, 1988, pp. 33-42.

[Sc87] B. Schieber, *Design and Analysis of Some Parallel Algorithms,* Ph. D. thesis, Tel Aviv University, Tel Aviv, Israel, 1987.

[SV88] B. Schieber, U. Vishkin, "On finding lowest common ancestors: simplification and parallelization," *Proc. 3rd Aegean Workshop on Computing,* Springer-Verlag LNCS 319, 1988, pp. 111-123.

[Ta72] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Computing,* vol. 1, 1972, pp. 146-160.

[Ta83] R. E. Tarjan, *Data Structures and Network Algorithms,* SIAM Press, Philadelphia, PA, 1983.

[TV84] R. E. Tarjan, U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM J. Computing,* vol. 14, 1984, pp. 862-874.

[Tu66] W. T. Tutte, *Connectivity in Graphs,* University of Toronto Press, 1966.

[Wh32] H. Whitney, "Non-separable and planar graphs," *Trans. Amer. Math. Soc.* 34, 1932, pp. 339-362.