In COMBINATORICA, vol. 12, 1992, pp. 53-76. (Figures not yet included in this file.)

A New Graph Triconnectivity Algorithm and Its Parallelization*

Gary L. Miller†
University of Southern California, Los Angeles, CA 90089

Vijaya Ramachandran‡
University of Texas, Austin, TX 78712

(Revised November 1990)

ABSTRACT

We present a new algorithm for finding the triconnected components of an undirected graph. The algorithm is based on a method of searching graphs called ‘open ear decomposition’. A parallel implementation of the algorithm on a CRCW PRAM runs in $O(\log^2 n)$ parallel time using $O(n+m)$ processors, where n is the number of vertices and m is the number of edges in the graph.

*A preliminary version of this paper was presented at the *19th Annual ACM Symposium on Theory of Computing,* New York, NY, May 1987.
† Supported by NSF Grant DCR 8514961.
‡ Supported by NSF Grant ECS 8404866 and the Semiconductor Research Corporation Grant 86-12-109.

1. Introduction

An *open ear decomposition* [Wh, Lo] of an undirected graph is a partition of its edge set into an ordered collection of paths called *ears* that satisfy certain properties. In this paper we present an efficient parallel algorithm based on open ear decomposition for testing vertex triconnectivity of an undirected graph and for finding the triconnected components of the graph. Our algorithm runs in $O(\log^2(n+m))$ parallel time using $O(n+m)$ processors on a CRCW PRAM, where n is the number of vertices in the graph and m is the number of edges. A sequential linear-time algorithm for the problem is available in Hopcroft & Tarjan [HoTa], but it is based on depth first search, and is not known to be efficiently parallelizable. Finding triconnected components of a graph is important in determining the connectivity structure of the graph and is useful in algorithms for determining planarity and for determining if two planar graphs are isomorphic.

Parallel NC algorithms for testing triconnectivity and for finding triconnected components are reported in Ja'Ja' & Simon [JaSi] and Miller & Reif [MiRe] but neither match our processor bound for general graphs. Ja'Ja' & Simon [JaSi] give an NC algorithm for finding triconnected components using $M(n)$ processors, where $M(n)$ is the number of processors needed to multiply two nxn matrices in polylog(n) time; currently $M(n) = O(n^{2.376})$. Miller & Reif [MiRe] present another algorithm for the problem that runs in $O(\log n)$ time on a CRCW PRAM with $n^{O(1)}$ processors. In contrast to these results we present an NC algorithm for the problem that uses only a linear number of processors. In particular, ours is the first efficient parallel algorithm for graph triconnectivity (see section 2 for the definition of an 'efficient parallel algorithm'). Our algorithm uses an efficient parallel algorithm for finding an open ear decomposition that we developed earlier in Miller & Ramachandran [MiRa] (see also [MaScVi]). More recently, building on the results we present, Ramachandran & Vishkin [RaVi] have obtained an efficient parallel triconnectivity algorithm that runs in logarithmic time. Also, Kanevsky & Ramachandran [KanRa] have used open ear decomposition to obtain better sequential and parallel algorithms for graph four connectivity.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the PRAM model. Section 3 gives graph-theoretic definitions; we warn the reader that there are a large number of definitions in this section. Section 4 gives a brief description of the main technical results leading to our triconnectivity algorithm. Section 5 establishes these results and section 6 gives an efficient implementation of the algorithm developed in section 5. Finally, section 7 extends these results to obtain an algorithm for finding the triconnected components of an input graph.

2. Model of Parallel Computation

The model of parallel computation that we will be using is the *PRAM* model, which consists of several independent sequential processors, each with its own private memory, communicating with one another through a global memory. In one unit of time, each processor can read one global or local memory location, execute a single RAM operation, and write into one global or local memory location.

PRAMs are classified according to restrictions on global memory access. An EREW PRAM is a PRAM for which simultaneous access to any memory location by different processors is forbidden for both reading and writing. In a CREW PRAM simultaneous reads are allowed but no simultaneous writes. A CRCW PRAM allows simultaneous reads and writes. In this case we have to specify how to resolve write conflicts. We will use the ARBITRARY model in which any one processor participating in a concurrent write may succeed, and the algorithm should work correctly regardless of which one succeeds. Of the three PRAM models we have listed, the EREW model is the most restrictive, and the ARBITRARY CRCW model is the most powerful. Any algorithm for the ARBITRARY CRCW PRAM that runs in parallel time T using P processors can be simulated by an EREW PRAM (and hence by a CREW PRAM) in parallel time $T \log P$ using the same number of processors, P (see, e.g., [KarRa]).

Let S be a problem which, on an input of size n, can be solved on a PRAM by a parallel algorithm in parallel time $t(n)$ with $p(n)$ processors. The quantity $w(n) = t(n) \cdot p(n)$ represents the *work* done by the parallel algorithm. Any PRAM algorithm that performs work $w(n)$ can be converted into a sequential algorithm running in time $w(n)$ by having a single processor simulate each parallel step of the PRAM in $p(n)$ time units. More generally, a PRAM algorithm that runs in parallel time $t(n)$ with $p(n)$ processors also represents a PRAM algorithm performing $O(w(n))$ work for any processor count $P < p(n)$.

Define $polylog(n) = \bigcup_{k>0} O(\log^k n)$. Let S be a problem for which currently the best sequential algorithm runs in time $T(n)$. A PRAM algorithm A for S, running in parallel time $t(n)$ with $p(n)$ processors is *efficient* if

a) $t(n) = polylog(n)$; and

b) the work $w(n) = p(n) \cdot t(n)$ is $T(n) \cdot polylog(n)$.

An efficient parallel algorithm is one that achieves a high degree of parallelism and comes to within a polylog factor of optimal speed-up with respect to the current best sequential algorithm. A major goal in the design of parallel algorithms is to find efficient algorithms with $t(n)$ as small as possible. The simulations between the various PRAM models make the notion of an efficient algorithm invariant with respect to the particular PRAM model used.

For the problem of testing triconnectivity and finding the triconnected components of a graph, the fastest sequential algorithm currently known runs in $O(m+n)$ time [HoTa], and this is also the best possible to within a constant factor. Hence the efficient parallel algorithm that we develop in this paper for this problem is the best possible, to within a polylog factor.

For more on the PRAM model and PRAM algorithms, see [KarRa].

3. Graph-theoretic Definitions

An *undirected graph* $G=(V,E)$ consists of a *vertex set* V and an *edge set* E containing unordered pairs of distinct elements from V. A *path* P in G is a sequence of vertices $< v_0, \cdots, v_k >$ such that $(v_{i-1}, v_i) \in E, i = 1, \cdots, k$. The path P *contains* the vertices $v_0, \cdots, v_k$ and the edges $(v_0, v_1), \cdots, (v_{k-1}, v_k)$ and has *endpoints* v_0, v_k, and *internal vertices* $v_1, \cdots, v_{k-1}$. The path P is a *simple path* if $v_0, \cdots, v_{k-1}$ are distinct and $v_1, \cdots, v_k$ are distinct. P is a *simple cycle* if it is a simple path and $v_0 = v_k$. Vertices v_i and v_j are *adjacent* on P if $i = j+1$ or $j = i+1$ and are *nonadjacent* otherwise.

We will sometimes specify a graph G structurally without explicitly defining its vertex and edge sets. In such cases, $V(G)$ will denote the vertex set of G and $E(G)$ will denote the edge set of G.

Let $P =< v_0, \cdots, v_{k-1} >$ 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 $< v_i, v_{i+1}, \cdots, v_j >$, if $i \le j$ or the path $< v_j, v_{j+1}, \cdots, v_i >$, 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.

An *ear decomposition* [Lo,Wh] $D = [P_0, \cdots, P_{r-1}]$ of an undirected graph $G=(V,E)$ is a partition of E into an ordered collection of edge disjoint simple paths $P_0, \cdots, P_{r-1}$ called *ears,* such that P_0 is a simple cycle and for $i > 0$, P_i is a simple path (possibly a simple cycle) with each endpoint contained in a smaller numbered ear, and with no internal vertices contained in smaller number ears. D is an *open ear decomposition* if none of the $P_i, i = 1, \cdots, r-1$ is a simple cycle. A *trivial ear* is an ear containing a single edge.

Let $D = [P_0, \cdots, P_{r-1}]$ 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 in E, we denote by $ear(e)$, the index of the unique ear that contains e. A vertex v will *belong to* $P_{ear(v)}$.

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 $G'' = (V', E'')$ where $E'' = E \cap \{(v_i, v_j) | v_i, v_j \in V'\}$.

An undirected graph $G = (V, E)$ is connected if there exists a path between every pair of vertices in V. For a graph G that is not connected, a *connected component* of G is a maximal induced subgraph of G which is 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. G is *biconnected* if it contains no cutpoint. A *biconnected component* of G is a maximal induced subgraph of G which is biconnected.

Let $G = (V, E)$ be a biconnected undirected graph. A pair of vertices $v_1, v_2 \in V$ is a *separating pair* for G if the induced subgraph on $V - \{v_1, v_2\}$ is not connected. G is *triconnected* if it contains no separating pair. We define *triconnected components* at the end of this section.

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 (see, e.g., [Ev]): 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 $V_B = K \bigcup$ {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 nonattachment vertices of B.

Let $G = (V, E)$ be a biconnected graph, and let Q be a subgraph of G. We define the *bridge graph of Q,* $S = (V_S, E_S)$ as follows: Let the bridges of Q in G be $B_i, i = 1, \cdots, k$. Then $V_S = V(Q) \bigcup \{B_1, \cdots, B_k\}$ and $E_S = E(Q) \bigcup \{(v, B_i) | v \in V(Q), 1 \le i \le k$, and v is an attachment of $B_i\}$.

Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. We will denote the bridge graph of ear P_i by C_i. An *internal attachment on* P_i is an edge of $G - P_i$ that is incident on a vertex of P_i other than its endpoints. Let the bridges of P_i in G that contain nonattachment vertices belonging to ears numbered lower than i be $B_1, \cdots, B_l$. We shall call these the *anchor bridges* of P_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. For ear P_0, we pick an edge (u, v) on P_0 as the *base edge* of P_0. For a pair of vertices a, b on P_0, $V_0(a, b)$ will be the vertices on the path from a to b on P_0 that avoids the base edge (u, v), excluding vertices a and b, and $V_0[a, b]$ will be the vertices on the path between a and b that contains

(u, v), excluding a and b.

Figure 1 illustrates some of our definitions relating to bridges.

Let $G = (V, E)$ be a graph and let P be a simple path in G. If each bridge of P in G contains exactly one vertex not on P, then we call G the *star graph of* P and denote it by $G(P)$. We denote the bridges of $G(P)$ by *stars,* i.e., a *star* is a connected graph in which at most one vertex has degree greater than 1. The unique vertex of a star that is not contained in P is called its *center.* Note that, in a connected graph G, the bridge graph of any simple path in G is a star graph. We will sometimes refer to a star graph $G(P)$ by G if the path P is clear from the context.

Two stars S_j and S_k in a star graph $G(P)$ *interlace* (see, e.g., [Ev, p. 149]) 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 belong to $S_j(S_k)$ and b and d belong to $S_k(S_j)$; or

2) There are three distinct vertices on P that belong to both S_j and S_k.

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 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 P is not a simple cycle and 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. (A star graph $G(P)$ has a planar embedding with P on the outer face if and only if no pair of stars interlace (see, e.g., [Ev, p. 150]).)

Let G be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Let $B_1, \cdots, B_l$ 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; and b) removing any two-attachment bridge with the endpoints of the ear as attachments. We will

G with open ear decomposition $D = [P_{0,} P_{1,} P_2, \quad P_3, \quad P_4]$; $P_0 =< a, b, c, d, e, a >$, $P_1 =< c, g, f, e >$, $P_2 = < d, f >$, $P_3 =< g, h, f >$, $P_4 =< c, i, e >$.

Bridges of P_1.

Bridge graph of P_1.

Ear graph G_1.

figure 1

Illustrating the definitions

call the star obtained by coalescing all anchor bridges, the *anchoring star* of $G_i(P_i)$. Figure 1 gives an example of an ear graph.

We conclude our list of definitions by defining the *triconnected components* of a biconnected graph (see, e.g., [Tu, HoTa2]). A *multigraph* $G = (V, E)$ is an undirected graph in which there can be several edges between the same pair of vertices. An edge e in a multigraph is denoted by (a, b, i) to indicate that it is the ith edge between a and b; the third entry in the triplet may be omitted for one of the edges between a and b.

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. If G has no separating pairs then G is triconnected. The pair a, b is a *nontrivial* separating pair if there are two nontrivial bridges of a, b in G.

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 $(V - V(X)) \bigcup \{a, b\}$. 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 $B \bigcup \{(a, b, i)\}$ and G_2 is $\bar{B} \bigcup \{(a, b, i)\}$. 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.

4. 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 the next section 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 ear P_i in D, form the coalesced graph of its ear graph and extract separating pairs from its star embedding. Section 6 provides an efficient parallel implementation for each step in the algorithm (with the exception of the construction of an open ear decomposition, for which an efficient parallel algorithm can be found in Miller & Ramachandran [MiRa] or Maon, Schieber & Vishkin [MaScVi]).

Section 7 builds on these results to give an efficient parallel algorithm to find the triconnected components of a graph. We find the triconnected components using Tutte splits in contrast to the earlier algorithm based on depth first search [HoTa].

5. Ear Decomposition and Triconnectivity

Lemma 1 [Wh] An undirected graph has an open ear decomposition if and only if it is biconnected.

Lemma 2 Let $D = [P_0, \cdots, P_{r-1}]$ 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 $H = \bigcup_{j=0}^{i-1} P_j$. 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 1. 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.[]

Lemma 3 Let $G = (V, E)$ be a biconnected undirected graph for which vertices x and y form a separating pair. Let $\mathbf{D}$ be an open ear decomposition for G. Then there exists an ear P_i in $\mathbf{D}$ 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 first ear P_0 contains no vertex in X_2 (see figure 2):

figure 2

Case 1 in the proof of Lemma 3

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_0 contains a vertex in X_2:

If P_0 contains no vertex in X_1, then case 1 applies to X_1. Otherwise P_0 contains at least one vertex from X_1, and one vertex from X_2. But then, since P_0 is a simple cycle, it must contain x and y, and again (by the argument of Case 1), every path from a vertex in $V_0(x, y)$ to a vertex in $V_0[x, y]$ must contain either x or y, and x and y are not adjacent on P_0.[]

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, 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 1, the pair c, e is a pair separating ears P_0 and P_4).

Lemma 4 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. 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 $P =< a, w_1, \cdots, w_l, b >$ 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 $P' =< w_r, \cdots, w_s >$ 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 4 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 internal attachment on P_i.
Proof Let x, y separate P_i. By Lemma 4, 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 internal attachment on P_i.

Conversely, suppose no anchor bridge of P_i has an internal attachment on P_i. 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 4, 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 $(i > 0)$. 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 4 and is omitted.

Claim 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 1 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. 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 4 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 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 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 $B_1, \cdots, B_k$ be the bridges of P_i in $C_i(P_i)$ corresponding to the anchor bridges of P_i in G. By Lemma 2, 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, $j = 1, \cdots, k$ 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 4, 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 5 Let $G(P)$ be a star graph with stars $S_1, \cdots, S_k$. For $j = 1, \cdots, k$ 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^*, $j = 1, \cdots, k$.
Proof Let x, y be a candidate pair for P. If x and y are the endpoints of P then the result follows. 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 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 $< a_1, \cdots, a_l >$ 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 $1 \le p < k$. 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 1 there exists a star S_j with an attachment a in $P[x, y] - \{x, y\}$ and an attachment 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 5 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.

We observe here that the coalesced graph $G_c(P)$ of a star graph $G(P)$ is unique. We omit the proof of this result because it is fairly straightforward but tedious. Further we do not exploit this uniqueness in the following except that we refer to G_c as ‘the’ coalesced graph of G (rather than ‘any’ coalesced graph of G).

Theorem 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 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 1.[]

Corollary to Theorem 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 5 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 $G_c{}^*$ as a *candidate set for* P. A pair of vertices is a candidate pair for P if and only if it lies in a candidate set for P. A candidate set S for ear P is a *nontrivial candidate set* if it contains a pair separating P.

Let G be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Since every separating pair for G is a candidate pair for some nontrivial ear P_i (Lemma 3), any algorithm that determines the candidate sets 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 sets in G by forming the bridge graph for each nontrivial ear, and then extracting the nontrivial candidate sets 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.

Lemma 6 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 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, and deleting multiple two-attachment bridges with the endpoints of the ear as attachments.

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 2, 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 1.[]

Lemma 7 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$, 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 sets in a biconnected graph.

Algorithm 1 Finding the Nontrivial Candidate Sets

Input A biconnected graph $G = (V, E)$.

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

2. *For* each nontrivial ear P_j *do*

A) Construct the ear graph $G_j(P_j)$.

B) Coalesce all interlacing stars on $G_j(P_j)$ to form the coalesced graph G_{j_c}. Construct the star embedding of ${G_{j_c}}^*$ of G_{j_c}, and identify each set of vertices on P_j on a common inner face in this embedding as a candidate set.

C) *If* $j > 0$ let the endpoints of P_j be u and v *else* let (u, v) be the base edge of P_0. *If* the anchoring star of P_j has an internal attachment on P_j, or if $j = 0$ then delete the candidate set $\{u, v\}$, if it exists.

D) Delete any doubleton candidate set for P_j that contains the endpoints of an edge in P_j.

rof;

Let $|V| = n$ and $|E| = m$. Step 1 has an $O(\log m)$ time parallel algorithm with $O(m)$ processors on a CRCW PRAM [MaScVi, MiRa]. In the next section, we give an $O(\log^2 m)$ time parallel algorithm on a CRCW PRAM with a linear number of processors for steps 2A and 2B. Clearly steps 2C and 2D are trivial to implement. Finally in section 7, we show how to obtain the triconnected components of a biconnected graph, given the nontrivial candidate sets.

6. Finding Candidate Sets

Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Let $|V| = n$ and $|E| = m$. In this section we give efficient parallel algorithms to implement steps 2A and 2B in Algorithm 1.

6.1. Forming the Ear Graphs

We give a divide and conquer algorithm for finding the ear graph for each nontrivial ear. Roughly speaking, the algorithm works as follows. Assume r is a power of 2. The algorithm has $\log r$ stages. In the first stage the algorithm computes the 'ear graph' of the subgraph of G consisting of the first $r/2$ ears and also the the 'ear graph' of the last $r/2$ ears. In general in the ith stage the algorithm computes the 'ear graph' of each subgraph of G consisting of the jth block of $r/2^i$ ears, $j = 1, \cdots, 2^i$. Thus, in the final stage the algorithm computes the ear graph of each ear.

In section 3 we defined 'ear graph' only with respect to a single ear. We now extend this definition to a collection of ears of the form $P_i, P_{i+1}, \cdots, P_j$. Given $G = (V, E)$ with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$, we define $G_{i,j}$, the (i, j) *ear graph of G, for i < j,* as follows: Let $P_{i,j} = \bigcup_{k=i}^{j} P_k$, and let $U_{i,j}$ be the set of vertices in $P_{i,j}$ that are contained in ears numbered lower than i (these are some of the endpoints of $P_i, P_{i+1}, \cdots, P_j$). Let $S_1, \cdots, S_k$ be the bridges of $P_{i,j}$ whose attachments are all in $U_{i,j}$, and among the remaining bridges of $P_{i,j}$ let $R_1, \cdots, R_l$ be the bridges that either contain a nonattachment vertex on an ear numbered lower than i, or contain at least 3 distinct attachments on $U_{i,j}$ (the *anchor bridges* of $P_{i,j}$). Let $C_{i,j}$ be the bridge graph of $P_{i,j}$. Then $G_{i,j}$ is the graph $C_{i,j}$ with the bridges corresponding to $R_1, \cdots, R_l$ coalesced (the *anchoring star of* $C_{i,j}$), and with the bridges $S_1, \cdots, S_k$ deleted. Note that $G_{i,i}$ is simply the ear graph G_i, and $G_{0,r-1}$ is the input graph G.

Let $G'_{i,j}$ be a graph consisting of $P_{i,j}$, together with a collection of stars with attachments on $P_{i,j}$, of which a subset $\{R_k, k = 1, 2, \cdots, l\}$ are 'marked'. Then $G'_{i,j}$ is a *partial ear graph of* $P_{i,j}$ if the graph obtained by coalescing the $R_k, k = 1, 2, \cdots, l$ is the ear graph $G_{i,j}$, excluding the attachments of the anchoring star to vertices in $U_{i,j}$. In our algorithm for finding the ear graphs we will mark stars that correspond to anchor bridges formed in the intermediate stages of the algorithm.

The following algorithm constructs the ear graph of each ear in parallel using the divide and conquer approach we outlined earlier. The construction proceeds in stages. In each stage the algorithm constructs a partial ear graph $G'_{i,j}$ for a collection of $P_{i,j}$'s. Finally, after having obtained $G'_{i,i}$ for each i, the algorithm coalesces all of the marked stars in $G'_{i,i}$ and provides attachments for this coalesced star to the two endpoints of P_i, to form the ear graph G_i.

Algorithm 2A Forming the Ear Graphs $G_i = (V_i, E_i), i = 0, \cdots, r-1$.

Input: Undirected biconnected graph $G = (V, E)$ with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$.

0. $G'_{0,r-1} \leftarrow G$.

1. *For* $i = 1, \cdots, \lceil \log r \rceil$ *do*

   for $j = 0$ (step 2) *to* $2^i - 2$ *pardo*

   let $a = \left\lceil \frac{jr}{2^i} \right\rceil, b = \left\lceil \frac{(j+1)r}{2^i} \right\rceil, c = \left\lceil \frac{(j+2)r}{2^i} \right\rceil$.

   a) Form $G'_{a,b-1}$ from $G'_{a,c-1}$ as follows:

   i) Delete the subgraph induced by $P_{a,b-1}$ from $G'_{a,c-1}$. Call the resulting graph $H_{b,c-1}$.

   ii) Find connected components in $H_{b,c-1}$.

   iii) Mark any connected component that contains a vertex that was previously marked.

   iv) Mark any connected component containing a vertex on an ear numbered lower than a (these will be some of the endpoints of ears in $P_{b,c-1}$) or containing 3 distinct attachments on $U_{a,b-1}$.

   v) Collapse each connected component into a single vertex.

   vi) Restore $P_{a,b-1}$ together with all edges incident on it. Remove multiple copies of any edge in this graph, and delete all vertices not connected to $P_{a,b-1}$.

   vii) Remove the bridges of $P_{a,b-1}$ with attachments only to vertices in $U_{a,b-1}$.

   viii) Remove any edge connecting a marked vertex to a vertex in $U_{a,b-1}$.

b) Form $G'_{b,c-1}$ from $G'_{a,c-1}$ in a similar manner.

rof

rof;

2. For each i, coalesce all marked bridges in $G'_{i,i}$ to form anchoring star S_i, and introduce attachments from S_i to the two endpoints of P_i.

Lemma 8 In step 1 of Algorithm 2A, for each i, the size of all of the $G'_{j,k}$ present in this step is $O(m)$.

Proof The analysis is similar to that in Lemma 7. Note that the size of all of the $G'_{j,k}$ excluding attachments to the corresponding $U_{j,k}$ is $O(n+m)$, since any edge of G appears at most twice in this collection. We now bound the number of attachment edges on $U_{j,k}$ in $G'_{j,k}$ over all j, k for fixed i.

Consider $G'_{j,k}$, where $j = y \cdot 2^i$ and $k = (y+1) \cdot 2^i$, for some j. First observe that in step vii we remove all bridges with attachments only on $U_{j,k}$, and for any marked vertex in $G'_{j,k}$ we remove all attachments to $U_{j,k}$ in step viii. So we only need to consider bridges that a) contain nonattachment vertices belonging only to ears numbered higher than k; b) have at most 2 attachments on $U_{j,k}$; and c) have at least 1 internal attachment on $G'_{j,k}$. For each such bridge, we can charge its (≤ 2) attachments on $U_{j,k}$ to an internal attachment and hence the total number of attachments on all of the $U_{j,k}$ is $O(n+m)$.[]

Lemma 9 Algorithm 2A correctly finds the ear graph of each ear.

Proof It is clear that Algorithm 2A without parts iii), iv), vii) and viii) in steps 1a and 1b, constructs the bridge graph of each ear. (Note that in this case every vertex will be connected to $P_{a,b-1}$ in step vi).) In the following we show that the algorithm as specified constructs the ear graph of each ear.

We establish this by showing by induction that in the ith iteration of the main step, each $G'_{j,k}$ formed is a partial ear graph of the corresponding $P_{j,k}$.

Base $i = 1$: Let $b = \lceil r/2 \rceil$. When $i = 1$ the algorithm constructs two graphs $G'_{0,b-1}$ and $G'_{b,r-1}$. Since the bridge graph $C_{0,b-1}$ of $P_{0,b-1}$ contains no anchor bridges, step 1a of Algorithm 2A constructs $C_{0,b-1}$, which is clearly a partial ear graph of $P_{0,b-1}$.

The bridge graph $C_{b,r-1}$ of $P_{b,r-1}$ contains only anchor bridges and all of these bridges have attachments only to $U_{b,r-1}$. Hence any partial ear graph of $P_{b,r-1}$ contains no bridges. Algorithm 2A removes all bridges of $P_{b,r-1}$ in step 1b vii) and hence the graph constructed by the algorithm is a partial ear graph of $P_{b,r-1}$.

Induction step Assume that the result is true until the $(i-1)$st iteration, and consider iteration i. Let $G'_{a,b-1}$ be constructed in step 1a from $G'_{a,c-1}$ in the ith stage.

First we show that any attachment v on $P_{a,b-1}$ of an anchor bridge of $P_{a,c-1}$ is also an attachment of an anchor bridge of $P_{a,b-1}$. Let the attachment edge be (v, w).

Case 1: Vertex v is an attachment of a bridge of $P_{a,c-1}$ that has a nonattachment vertex u on an ear numbered lower than a. The vertex v continues to be an attachment of a bridge of $P_{a,b-1}$ that contains u. But since $ear(u) < a$, v is an attachment of an anchor bridge of $P_{a,b-1}$.

Case 2: Vertex v is an attachment on $P_{a,b-1}$ of a bridge of $P_{a,c-1}$ that has at least 3 distinct attachments x, y, z on $U_{a,c-1}$; we allow the possibility that v is one of x, y, z. In this case x, y, z and edge (v, w) belong to a single bridge of $P_{a,b-1}$. If x, y and z are in $U_{a,b-1}$ then the same property holds in $P_{a,b-1}$. Otherwise assume x is not in $U_{a,b-1}$ (note that x cannot be v). Then since x is in $U_{a,c-1}$, we have $ear(x) < a$ and hence the bridge of $P_{a,b-1}$ that contains edge (v, w) is an anchor bridge of $P_{a,b-1}$.

Now we show that every internal attachment of the anchoring star of $G_{a,b-1}$ (i.e., all attachments except those in $U_{a,b-1}$) is contained in one of the marked stars of $G'_{a,b-1}$. Let v be an attachment vertex of the anchoring star of $P_{a,b-1}$ in G. If v is also an attachment of the anchoring star of $P_{a,c-1}$ in G then by the induction hypothesis there is a marked star in $G'_{a,c-1}$ that has an attachment to v. The connected component containing this marked star is marked in step 1a iii) of Algorithm 2A and hence v is an attachment of a marked star in $G'_{a,b-1}$. If v is not an attachment of an anchoring star of $P_{a,c-1}$ in G then in order for v to be an attachment edge of an anchoring star in $C_{a,b-1}$, either it must be connected to an anchoring star of $P_{a,c-1}$ through internal vertices in $P_{b,c-1}$ or it must be connected through internal vertices in $P_{b,c-1}$ to a vertex in $U_{b,c-1}$ that belongs to an ear numbered lower than a. In the former case the corresponding connected component is marked in step 1a iii) of Algorithm 2A and in the latter case it is marked in step 1a iv) of Algorithm 2A.

If an attachment does not belong to an anchoring star of $G_{a,b-1}$ or $G_{b,c-1}$ then we claim that it cannot be an attachment of a marked star in $G'_{a,b-1}$ or $G'_{b,c-1}$. This is so since the algorithm only deletes edges from the graph, and hence never induces a path between two vertices in a subgraph of $G'_{x,y}$ if a path did not exist in the subgraph of G induced by $V(G'_{x,y})$.

Finally we note that every non-anchor bridge of $P_{a,b-1}$ in G appears as a non-anchor bridge in $G'_{a,b-1}$ as constructed by Algorithm 2A. This is because steps 1a vii) and viii) and steps 1b vii) and viii) delete only edges incident on vertices belonging to ears numbered lower than a and b respectively, and hence none of the edges in non-anchor bridges are removed.

A similar argument holds for $G_{b,c-1}$.

This establishes the induction step and the lemma is proved.[]

The major computation in each parallel step of the algorithm involves finding connected components in subgraphs of the $G'_{i,j}$'s. Since by Lemma 8, the total size of the graphs present at any given step is $O(m)$, each parallel step can be implemented in $O(\log m)$ time on a CRCW PRAM with $O(m)$ processors [ShVi]. Finally since the algorithm has $\log m$ parallel stages, it runs on a CRCW PRAM in $O(\log^2 m)$ time with $O(m)$ processors.

6.2. Forming the Coalesced Graph

We now turn to step 2B, which finds the coalesced graph G_c of a star graph $G(P)$, and determines from it, the candidate sets of $G(P)$.

Our parallel algorithm to coalesce all interlacing stars in a star graph $G(P)$ runs in $O(\log^2 q)$ time using $O(q)$ processors, where q is the number of edges in the stars and in the path. We can consider the star embedding of a star graph with no interlacing stars to consist of embedding the path P as a horizontal line, and all the stars and their edges above the line in a well nested fashion with no two edges crossing. Each face in this embedding, ignoring the exterior face, lies directly below exactly one star and every star with k edges sits directly above $k-1$ faces. Thus, associated with the embedded stars is an ordered rooted tree: one vertex for each star and one for each face, which we call *star-vertex* and *face-vertex* respectively. The root of this tree is the exterior face. The children of a star-vertex are its faces in order and the children of a face-vertex are the stars that lie directly below it in order. We call this tree the *face-star tree.* The face-star tree is the main data structure that we use for the parallel star coalescing algorithm.

We distinguish between distinct and nondistinct attachments of the stars. In figure 3 we show a star embedding of a star graph $G(P)$ together with its face-star tree for the case of distinct attachments, i.e., each vertex on P is common to at most one star. As one can see from the example, the face-star tree is any rooted and ordered tree such that 1) the root is labeled a face, and each level of vertices is labeled alternately as faces and stars, and 2) each leaf is labeled as a face. The last condition just requires that all leaves are of even depth.

The face-star tree gives us most of the information we need about the star embedding. We show how to extract the information that is in the embedded star graph from the face-star tree. We start with a definition. The *Euler tour* of an ordered rooted tree T is the cycle starting at the root that traces the tree in a counter-clockwise order. In our example from figure 3 the Euler tour is $< A, a, B, a, C, c, D, c, C, d, E, d, C, a, A, b, F, b, G, e, H, e, G, f, \quad I, f, J, f, G, b, A >$, where face vertices are labeled by upper case letters and star vertices by lower case letters. A *corner* of an Euler tour $< v_1, \cdots, v_k >$ is any triple (v_{i-1}, v_i, v_{i+1}) of successive vertices on the tour. The triple is a corner of star (face) if v_i is a star (face). Note that each attachment is defined by a corner of the star it belongs to, with the two face vertices in the corner corresponding to the faces bounding the attachment edge.

Star embedding

Face-star tree

figure 3
A star embedding with distinct attachments
and its face-star tree

The general case of nondistinct attachments is slightly more complicated. We assume, without loss of generality, that $G(P)$ has no multiple two-attachment stars with the same pair of attachments, since otherwise we could delete the multiple copies without altering the candidate sets for P; for the same reason we also assume that $G(P)$ has no two-attachment star with the same span as another star with three or more attachments. Hence any star graph with a star embedding that is derived from $G(P)$ by coalescing stars has a unique embedding. Each attachment vertex corresponds to a consecutive set of star corners in the Euler tour of the face-star tree of such an embedding. For instance in figure 4 we exhibit a star embedding in which the attachments are not all distinct. Here the attachment x corresponds to the corners (F, c, B), (B, a, A), (A, d, G) and (G, e, I). Thus, since our algorithm will maintain the face-star tree and not the star graph, we need to maintain a list of corners that determine the beginning and end of each attachment vertex.

figure 4

A general star embedding with its face-star tree

Our parallel star coalescing algorithm is a divide-and-conquer algorithm that divides the set of stars in half and recursively and in parallel finds the face-star tree for the coalesced graph in each half, and then combines the two trees using *Merge-Tree,* the heart of the algorithm. We give an $O(\log p)$ time p processor algorithm to merge the face-star trees of the two halves, where p is the total number of attachments in the two two coalesced graphs. This gives an $O(\log^2 q)$ time q processor algorithm for star coalescing.

The procedure Merge-Tree is substantially simplified if we first sort the original stars as follows. For a star S let $left(S)$ and $right(S)$ be the first and last attachments of S on P and let *span* of S be the closed interval $[left(S), right(S)]$. We say a vertex u on P is less than another vertex v on P if u appears before v on P. We say that star S is before star S' in the *Euler order* if (1) $left(S) < left(S')$ or (2) $left(S) = left(S')$ and $right(S) > right(S')$. Note that in the star embedding, the Euler order gives the sequence in which the stars first appear on the Euler tour.

We first sort the stars in $G(P)$ with respect to the Euler order in $O(\log q)$ time using q processors [Co] (as a preprocessing step we coalesce all stars with the same span, -- note that any such pair must interlace). By the following observation it is relatively easy to maintain the stars in order even if they are being coalesced: If interlacing stars are coalesced, then the Euler order on the new stars is obtained from the old Euler order by taking each interlacing class of stars and associating it with the minimum numbered star in its class. Thus we can maintain the Euler order throughout the computation without sorting.

The main procedure Merge-Tree has as input the face-star trees, T and T' respectively, of two star embeddings $G^*(P)$ and $G'^*(P)$. By presorting, as defined above, we may also assume that there is a point δ such that every star in $G(P)$ has a point of attachment at δ or to its left, and every star in $G'(P)$ has all of its attachments at δ or to its right. The procedure returns with the face-star tree of the star embedding of the coalesced graph of $G(P) \cup G'(P)$. The procedure has four steps which may be combined in a slightly more efficient algorithm, but for clarity, are best viewed separately:

1) Determine which stars in T (T') interlace with the same star in T' (T).

2) Coalesce all stars in T (T') that interlace with the same star in T' (T).

3) For each pair of interlacing stars S, S', with $S \in T$ and $S' \in T'$, add the attachments of S to the right of δ to those of S'.

4) Splice the tree T' into the tree T.

To implement step 1 we start by determining those stars in T' that interlace with the same star in T. We use the fact that every star S in T has an attachment at δ or to its left. Note that if S' is a star in T' that interlaces with S, then S interlaces with the parent star of S', if it exists. Thus, if $S'_1, \cdots, S'_k$ are the stars that are the children of the root of T', and S' is a descendant of S'_i, then S must interlace with every star on the path from S' to S'_i in T'. Thus it suffices to find, for each attachment of each star S in T, the lowest level star in T' with which it interlaces. We can find this star by determining in which corner of T' the attachment lies. We do this by merging the attachments of $G(P)$ with those of $G'(P)$, either by sorting the points or by using a relatively simple pointer jumping scheme. This step can be preformed in logarithmic time with a linear number of processors.

To find the set of stars of T that interlace with a given star in T', we note that the stars of T that interlace with some star in T' all lie on the rightmost path, say τ, in T, i.e., the path that starts from the root and ends at the rightmost leaf. Each star S' in T' interlaces with exactly those stars in T that have an attachment in the span of S', excluding the endpoints; this corresponds to stars on a single segment of τ. Thus each star in T' can determine this segment of τ in constant time given our preprocessing.

We now have the set of stars in T that must be coalesced and the set of stars in T' that must be coalesced. We do this coalescing in step 2. Clearly we can coalesce two neighboring stars in constant time, since one star must be the parent of the other, or the two stars must be siblings. Thus this step can be done in $O(\log p)$ time with $O(p)$ processors using either parallel tree contraction [MiRe] or the Euler tour technique on trees [TaVi].

At this point at most one star in T' can interlace with a given star in T and at most one star in T can interlace with a star in T'. Let us call a star that currently interlaces with a star in the other half, an *interlacing star.* The only interlacing stars in T lie on the path τ, and the only interlacing stars in T' are children of the root. In step 3 we add the attachments to the right of δ of each interlacing star in T to its mate in T' so that the interlacing stars in T' have their own attachments as well as the attachments (to the right of δ) of their mate in T.

Finally in step 4 we replace all subtrees to the right of δ for each interlacing star S in T with all the subtrees of its mate S'_j in T'. This can be performed in constant time. On the other hand, if S'_i is a star in T' that does not interlace with any star in T then it must lie in one of the faces defined by T. In this case we attach the subtree at S'_i to this face.

This concludes the description of the algorithm to coalesce all interlacing stars in a star graph with q edges in $O(\log^2 q)$ time using $O(q)$ processors. Since the total size of the ear graphs of all nontrivial ears is $O(m)$ this gives an algorithm to find the coalesced graphs of all ear graphs of G in $O(\log^2 n)$ time using $O(m)$ processors.

The face-star tree data structure now allows us to extract the candidate sets efficiently. For each face vertex f, the information available at its parent in the tree gives the leftmost vertex l and rightmost vertex r on P that belong to the face f. Similarly, the information at each child vertex c gives an open interval s_c on P between l and r that does not belong to face f. The candidate set defined by f is the set of vertices in the interval $[l, r]$ excluding the vertices in the intervals s_c, where c ranges over the children of f in the face-star tree. Thus the vertices on each face in the star embedding of the coalesced graph can be obtained as a circular linked list in constant time by having a processor at each vertex in the face-star tree. This gives the candidate sets for P.

7. Finding Triconnected Components

We start by defining a special type of split, called an *ear split,* on a biconnected graph with an open ear decomposition. Let G be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Let a, b be a pair separating ear P_i. Let $B_1, \cdots, B_k$ be the bridges of P_i with an attachment in $V_i(a, b)$, and let $T_i(a, b) = (\bigcup_{j=1}^{k} B_j) \bigcup P_i(a, b)$. 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* $G_1 = T_i(a, b) \bigcup \{(a, b, i)\}$ and the *lower split graph* $G_2 = \bar{T}_i(a, b) \bigcup \{(a, b, i)\}$. 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 set for ear P_i. A pair u, v in S is an *adjacent separating pair for* P_i if u, v is a pair separating P_i and S contains no vertex in $V_i(u, v)$. A pair a, b in S is an *extremal separating pair for* P_i if S contains no vertex in $V_i[a, b]$.

We now prove the following theorem.

Theorem 2 Let $G = (V, E)$ be a biconnected graph with an open ear decomposition

$D = [P_0, \cdots, P_{r-1}]$. 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 $P_j, 0 \le j \le r-1$ in G. If $\{c, d\} \ne \{a, b\}$ or $i \ne j$ 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 set 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 $j \ge i$, that the graph $P_{0,j} = \bigcup_{k=0}^{j} P_k$ satisfies the property in part b) of the Theorem. The details are straightforward and are omitted.

c and d) If $i \ne j$ 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 Finding Triconnected Components

Input A biconnected graph $G = (V, E)$ with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$, and the nontrivial candidate sets for each ear.
Output The triconnected components of G.

1. *For* each nontrivial ear P_i do

   for each nontrivial candidate set S for P_i *do*

   a) *For* each adjacent separating pair u, v in S 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 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 2.

   b) *If* $|S| > 2$, *then* 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 2 and replace D by D_2. (If $i = 0$ and (u, v) is the base edge of P_0 then perform this ear split only if there are at least two edges between u and v.)

   rof

   rof;

2. Split off multiple edges in the remaining split graphs to form the bonds.

Lemma 10 Algorithm 3 generates the Tutte components of G.
Proof By Theorem 2, each split performed in Algorithm 3 is a Tutte split, and at termination there is no separating pair in any of the generated graphs.[]

We now consider a parallel implementation of Algorithm 3. First consider one iteration of the outer for loop in step 1: the algorithm performs all of the ear splits for a given ear P_i in such an iteration. By Theorem 2 these ear splits can be performed in any order. Consider a listing $L = [< a_1, b_1 >, \cdots, < a_k, b_k >]$ of the adjacent pairs separating P_i, with each $a_i < b_i$, and with the pairs sorted in nondecreasing order of the a_i, ties being resolved in decreasing order of the b_i. We number the corresponding ear splits in order as $1, \cdots, k$. Each of these splits creates a new connected graph, and hence after processing all of these splits we will have $k + 1$ connected graphs. Let G_i be the graph obtained from the upper split graph formed at the ith split (with possibly some upper split graphs split off from it by later splits); G_0 is the initial graph with all of the upper split graphs split off from it.

To determine the end result of performing step 1a on all nontrivial candidate sets of ear P_i it suffices to determine for each edge incident on a vertex in P_i, the graph G_i that contains it. For this we compute the bridges of P_i. For each bridge B (it suffices to consider only bridges with an internal attachment on P_i) we compute its span $[a, b]$. If there is no pair $< x, y >$ in L with $x \le a$ and $y \ge b$ then we assign B to G_0. Otherwise we associate with B a pair $< x, y >$ in L as follows. If B has three or more attachments we find the last pair $< x, y >$ in L with the property $x \le a$ and $y \ge b$. If B has exactly two attachments then $< x, y >$ is the last pair in L with the property that either $x < a$ and $y \ge b$ or $x \le a$ and $y > b$. Let $< x, y >$ be the jth pair in L. We assign the edges in bridge B to G_j. A similar computation assigns each edge on P_i to the appropriate G_j (we use the criterion for bridges with 3 or more attachments). Using the definition of an ear split it is straightforward to see that this computation assigns each edge to the correct G_j. Finally we place virtual edges between adjacent separating pairs in each G_j. All of this computation can be performed in logarithmic time with a linear number of processors using parallel algorithms for graph connectivity [CV] and sorting [Co]. Step 1b can be performed with similar bounds in the same manner.

In order to fully parallelize Algorithm 3 we need to perform the above computation in parallel for several nontrivial ears. Since we need to look at the edges incident on each ear that we are working with, we do not attempt to perform this computation for all ears in parallel, since an edge could be incident on several ears. Instead we break up the computation into $O(\log r)$ stages as shown below.

In the first stage of the parallel algorithm we locate the median *active* ear, call it P_i, where an *active ear* is an ear having a nontrivial candidate set. Then we find the bridges of $P_{0,i-1}$ (the union of ears P_0 to P_{i-1}), and in each bridge we locate the smallest-numbered active ear. Let these ears be $P_i = P_{i_0}, P_{i_1}, \cdots, P_{i_l}$. We compute the bridges of these ears by computing the bridges of $H = \bigcup_{j=0}^{l} P_{i_j}$.

Let L_0 be the lower split graph that remains when all of the ear splits for the ears $P_{i_j}, j = 0, \cdots, l$ have been performed. The graph L_0 contains the graph $P_{0,i-1}$. Since each of the P_{i_j} lies in a different bridge of $P_{0,i-1}$ it follows that any bridge of H that has an internal attachment on more than one of the P_{i_j} must be contained in L_0. Hence, in order to perform the parallel implementation of step 1 on P_{i_j} we only need to work with those edges incident on a vertex contained in P_{i_j} that either lie on P_{i_j} or belong to a bridge of H that contains internal attachments only on P_{i_j}. We perform this computation in parallel for each $P_{i_j}, j = 0, \cdots, l$. At this point we have a collection of split graphs, and by Theorem 2, the number of active ears in any of them is at most half the number of active ears in the original graph. We now recursively apply the above step in each of the split graphs, and in $\log r$ stages we will be done. This gives an $O(\log^2 m)$ time algorithm with $O(m)$ processors to find the triconnected components.

We also note that Algorithm 3 can be easily modified to obtain the tree of 3-connected components (or ‘auxiliary graph’ [HoTa2]) with the same time and processor bounds. For this we construct the tree of triconnected components by introducing a vertex for each copy of edge (a, b, i) added at a split and we place an edge between each pair of vertices whose edges were added at the same ear split. At the end of the algorithm, we identify, for each vertex in this auxiliary graph, the triconnected component in which it lies and we replace all vertices corresponding to the same triconnected component by a single vertex representing that component.

REFERENCES

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

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

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

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

[HoTa2] J. E. Hopcroft, R. E. Tarjan, "Finding the triconnected components of a graph," TR 72-140, Computer Science Department, Cornell University, Ithaca, NY, 1972.

[JaSi] J. Ja’Ja, J. Simon, "Parallel algorithms in graph theory: planarity testing," *SIAM J. Comput.,* 11, 1982, pp. 314-328.

[KanRa] A. Kanevsky, V. Ramachandran, "Improved algorithms for graph four-connectivity," *Proc. 28th IEEE Symp. on Foundations of Comp. Sci,* 1987, pp. 252-259; *Jour. Comput. Syst. Sci.,* to appear.

[KarRa] 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.

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

[MaScVi] 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.

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

[MiRa2] 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.

[MiRe] G. L. Miller, J. H. Reif, "Parallel tree contraction and its applications," *Proc. 26th IEEE Symp. on Foundations of Comp. Sci,* 1985, pp. 478-489.

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

[TaVi] R. E. Tarjan, U. Vishkin, "Finding biconnected components and computing tree functions in logarithmic parallel time," *SIAM J. Comput.,* vol. 14, 1985, pp. 862-874.

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

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