In J. Comp. Sys. Sci., FOCS'87 Special Issue, vol. 42, 1991, pp. 110-125. (Figures are not yet included in this file.)

IMPROVED ALGORITHMS FOR GRAPH FOUR-CONNECTIVITY

Arkady Kanevsky

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

Vijaya Ramachandran

Department of Computer Sciences
University of Texas
Austin, TX 78712

ABSTRACT

We present a new algorithm based on open ear decomposition for testing vertex four-connectivity and for finding all separating triplets in a triconnected graph. A sequential implementation of our algorithm runs in $O(n^2)$ time and a parallel implementation runs in $O(\log^2 n)$ time using $O(n^2)$ processors on an ARBITRARY CRCW PRAM, where n is the number of vertices in the graph. This improves previous bounds for the problem for both the sequential and parallel cases. The sequential time bound is the best possible, to within a constant factor, if the input is specified in adjacency matrix form, or if the input graph is dense.

1. Introduction

This paper deals with the problem of determining four connectivity in an undirected graph. Connectivity is an important graph property and there has been a considerable amount of work on algorithms for determining k-connectivity in graphs. An important application of this property is that a k-connected network can operate in a reliable manner in the presence of up to k node or link failures.

This research was supported by National Science Foundation under ECS 8404866, the Semiconductor Research Corporation under 86-12-109 and the Joint Services Electronics Program under N00014-84-C-0149 while both authors were with the Coordinated Science Laboratory, University of Illinois, Urbana, IL.
A preliminary version of this paper appeared in the *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science,* 1987 [KanRa].

There are well-known sequential linear-time algorithms for determining vertex connectivity and biconnectivity (see e.g., [Ev]), as well as triconnectivity [HoTa,MiRa2]. The best previously-published deterministic sequential algorithms for testing graph 4-connectivity have time complexity $O(nm)$, where n is the number of vertices in the input graph and m is the number of edges. There are two such algorithms. One is based on a reduction to network flow [EvTa, Ev2, Ga, GiSo]. The other uses the $O(m)$ algorithm for testing triconnectivity [HoTa, MiRa2] to test four-connectivity in a triconnected graph in $O(mn)$ time by deleting each vertex of the graph in turn, and testing triconnectivity in the resulting graph; this algorithm also finds all separating triplets in the graph, if the graph is not triconnected. For the problem of finding all separating k-sets, it is known that the number of separating k-sets in a k-connected graph is $O(n^2)$ for any fixed k [Ka]. We also note that there are some randomized algorithms for testing k-connectivity for $k > 3$ [BeX,LiLoWi]; the running time of these algorithms is $\Omega(n^{5/2})$.

In this paper we present a new sequential algorithm, based on open ear decomposition [Lo, MaScVi, MiRa, Wh], that tests vertex four-connectivity and finds all separating triplets in a triconnected graph in $O(n^2)$ time. This represents an improvement in the running time over all previous algorithms for the problem, both deterministic and probabilistic. We also present a parallel implementation of the algorithm, which runs in $O(\log^2 n)$ time using $O(n^2)$ processors on an ARBITRARY CRCW PRAM. For comparison the best previous processor count for an NC algorithm for this problem is $O(nm)$, which is obtained by running n parallel applications of the parallel triconnectivity algorithms in [MiRa2,RaVi] on the input graph with a vertex deleted.

Our algorithm thus gives improved performance bounds for both the sequential and parallel case. It also gives a completely new method for the four-connectivity problem, which is of interest in itself. We also note that the algorithm is easily modified to work for edge four-connectivity as well with the same time and processor bounds: we use an ear decomposition instead of an open ear decomposition. While a sequential $O(n^2)$ time algorithm is already known for edge four-connectivity [Ma], our algorithm gives the best processor count for an NC algorithm for edge four-connectivity. We do not elaborate further on this.

The rest of this paper is organized as follows. In section 2 we describe the model of parallel computation we use. Section 3 gives graph-theoretic definitions. Section 4 relates open ear decomposition to vertex four-connectivity, and gives a high-level description of the four-connectivity algorithm. Finally, in section 5, we show how to implement this algorithm in $O(n^2)$ sequential time, and in $O(\log^2 n)$ parallel time with n^2 processors on an ARBITRARY CRCW PRAM.

2. Model of Parallel Computation

The model of parallel computation that we will be using is the *Parallel Random Access Machine* or *PRAM* [KarRa], which consists of several independent sequential processors, each with its own private memory, communicating with one another through a global memory. In one step, each processor can read one global or local memory, 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 that, 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. 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 more on the PRAM model and PRAM algorithms, see [KarRa].

Our efficient parallel algorithm for four-connectivity works on an ARBITRARY CRCW PRAM. Some of the subroutines also work on the more restrictive EREW PRAM model within the same processor and time bounds.

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* $v_1, \cdots, v_{k-1}$.

Given a path $< v_0, \cdots, v_k >$, v_i is to the *left* of v_j and v_j is to the *right* of v_i if $i < j$. 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 $v_0 = v_k$. A single vertex is a trivial path with no edges.

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.

Let $G = (V, E)$ be an undirected graph and let $V' \subseteq V$. A graph $G' = (V', E')$ is a *subgraph* of G if $E' \subseteq E \bigcap \{(v_i, v_j) | v_i, v_j \in V'\}$. The *subgraph of* G *induced by* V' is the graph $G'' = (V', E'')$ where $E'' = E \bigcap \{(v_i, v_j) | v_i, v_j \in V'\}$.

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 an *articulation point (a.p.)* or *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 articulation point.

Let $G = (V, E)$ be a biconnected undirected graph. G is triconnected if for all pairs of vertices $v_1, v_2 \in V$ the induced subgraph on $V - \{v_1, v_2\}$ is connected.

Let $G = (V, E)$ be a biconnected graph which is not triconnected. A *(nontrivial) separating pair* in G is a pair of vertices u, v in V whose removal decomposes G into two or more connected components. A *trivial separating pair* is a pair of vertices u, v with (u, v) an edge (note that a pair of vertices can be both a trivial and a nontrivial separating pair). A *candidate pair* is a trivial or nontrivial separating pair; a *candidate set* is a set of vertices such that each pair in the set is a candidate pair.

A triplet (v_1, v_2, v_3) of unordered distinct vertices in V is a separating triplet of a triconnected graph if the subgraph induced by $V - \{v_1, v_2, v_3\}$ is not connected. G is *four-connected* if it contains no separating triplet.

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 $P_0, \cdots, P_{r-1}$ such that P_0 is a simple cycle and each endpoint of $P_i, i = 1, \cdots, r-1$ is contained in some P_j, $j < i$, while none of its internal vertices are contained in any P_j, $j < i$. The paths in D are called the *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 consisting of a single edge. A graph has an open ear decomposition if and only if it is biconnected [Wh].

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. 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. The nontrivial and trivial bridges together form the bridges of Q in G.

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 graph and let P be a simple path in G. If each bridge of P in G contains exactly one vertex not on P, and there is a bridge B of P with the endpoints of P as attachments, then we call G the *star graph of* P and denote it by $G(P)$. We denote the bridges of P in $G(P)$ by *stars.* 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.

Let $G(P)$ be a star graph, and let $S_1, \cdots, S_k$ be some of the stars in $G(P)$. The operation of *coalescing* the stars $S_i, i = 1, \cdots, k$ removes these stars and replaces them by a new star S whose attachments are the union of the attachments of $S_1, \cdots, S_k$.

Let G be a biconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Let the bridges of P_i in G that contain non-attachment vertices on ears numbered lower than i be $B_1, \cdots, B_l$. We shall call these the *anchor bridges of* 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 coalescing all stars corresponding to anchor bridges, and by deleting multiple two-attachment bridges. We will call this coalesced star, the *anchoring star* of $G_i(P_i)$. For any two vertices x, y on P_i, we denote by $V_i(x, y)$, the internal vertices of $P_i(x, y)$; we denote by $V_i[x, y]$, the vertices in $(P_i[x, y] - \{x, y\}) \bigcup$ {vertices in the anchor bridges of P_i}. For a star graph $G(P)$ with no anchoring star, the set $V(x, y)$ represents the vertices in $P(x, y) - \{x, y\}$, and the set $V[x, y]$ represents the vertices in $P[x, y] - \{x, y\}$.

Figure 1 illustrates some of our definitions relating to bridges.

Two stars S_j and S_k in a star graph $G(P)$, where P is a simple path, *interlace* (see, e.g., [Ev, p. 149]) if one of the following two hold:

1) there exist four distinct vertices a, b, c, d in increasing order in 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.

Given a star graph $G(P)$, the *coalesced graph* G' of G is the graph obtained from G by coalescing all pairs of stars that interlace.

a) G with open ear decomposition $D = [P_0, P_1, P_2, P_3, P_4]$;
$P_0 =< 1, 2, 3, 4, 5, 1 >$, $P_1 =< 3, 7, 6, 5 >$, $P_2 = < 6, 4 >$, $P_3 =< 7, 8, 6 >$, $P_4 =< 3, 5 >$.

b) Bridges of P_1.

c) Bridge graph G_1 of P_1.

d) Ear Graph of P_1

figure 1
Illustrating the definitions

4. Open Ear Decomposition and Four-connectivity

Lemma 1 Let $G = (V, E)$ be a triconnected undirected graph for which $t = (x, y, z)$ forms a separating triplet. Let $\mathbf{D} = [P_0, \cdots, P_{r-1}]$ be an open ear decomposition for G and let $G_i(P_i)$ be the ear graph of ear P_i. Then there exists an ear P_i in $\mathbf{D}$ that contains two of the three vertices in t, say x and y, such that both $V_i(x, y)$ and $V_i[x, y]$ contain a vertex other than z, and every path from a vertex in $V_i(x, y)$ to a vertex in $V_i[x, y]$ in G_i passes through x, y or z. Further ear P_i uniquely determines a connected component C in the subgraph induced by $V - \{x, y, z\}$, in the sense that no other ear P_j in G that contains x, y and a vertex in C, has the property that $V_j(x, y) - \{x, y, z\}$ is nonempty, and every path between a vertex in $V_j(x, y)$ and a vertex in $V_j[x, y]$ in G_j contains x, y, or z.

Proof Since $t = (x, y, z)$ forms a separating triplet, the subgraph of G induced by $V - \{x, y, z\}$ contains at least two connected components. Let C_1 and C_2 be two such connected components.

Case 1 The first ear P_0 contains no vertex in C_2 (see figure 2):

figure 2

Case 1 in the proof of Lemma 1

Consider the lowest-numbered ear, P_i, that contains a vertex v in C_2. Since its endpoints are distinct and must be contained in lower-numbered ears, P_i must enter C_2 through one of the three vertices in t, say x, and must leave C_2 through one of the remaining two vertices in t, say y. Thus P_i must contain two of the three vertices in t, and $V_i(x, y)$ contains at least one vertex other than z. Further, all vertices in $V_i(x, y)$ lie in C_2, and none of the vertices in $V_i[x, y]$ lie in C_2. Thus the vertices in $V_i(x, y)$ are separated from the vertices in $V_i[x, y]$ by t.

To prove the second claim of the lemma for this case, let $C_2 = C$, and suppose P_j is an ear that contains x and y and also a vertex, say u, in C. Then $j > i$, since P_i is the lowest-numbered ear to contain a vertex in C. Since P_i contains x and y, x and y must be the endpoints of P_j, and all other vertices on it lie in $C \cup \{z\}$. Further, since $i < j$ and vertex v is contained in P_i, the vertices in the bridge of P_j containing v (call it B') are in $V_j[x, y]$, and since C is a connected component in the subgraph induced by $V - \{x, y, z\}$, there is a path from B' to the vertex u in $V_j(x, y)$ that avoids x, y, and z. This establishes the second claim of the lemma for this case.

Case 2 P_0 contains a vertex in C_2:

If P_0 contains no vertex in C_1, then case 1 applies to C_1. Otherwise P_0 contains at least one vertex in C_1 and one vertex in C_2. But then, since P_0 is a simple cycle, it must contain two of the three vertices in t, say x and y, such that (by the argument of case 1), every path from a vertex in $V_0(x, y)$ to a vertex in $V_0[x, y]$ contains x, y or z, and P_0 is the unique ear with this property, which has a vertex in C_2. Thus, by taking C_2 to be C, the lemma is established.[]

We will say that a separating triplet $t = (x, y, z)$ *separates* ear P_i if P_i contains two of the vertices in t, say x and y, with $V_i(x, y)$ not a subset of $\{z\}$, 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 $V - \{x, y, z\}$. We will denote this by writing t as $i([x, y], z)$ to indicate that P_i contains x and y, and $V_i(x, y)$, which contains a vertex other than z, is separated from $V_i[x, y]$ by $\{x, y, z\}$. By Lemma 1, every separating triplet in G separates some ear, and hence can be written in the above form. We will write $i([x, y], z)$ as simply $([x, y], z)$, if the ear number is obvious from the context.

Analogously, for a star graph $G(P)$, a triplet of vertices $t = ([x, y], z)$ in G *separates* P if P contains x and y, $V(x, y) - \{z\}$ and $V[x, y] - \{z\}$ are non-empty, and the vertices in $V(x, y)$ are separated from the vertices in $V[x, y]$ when x, y and z are deleted from $G(P)$.

Lemma 2 Let $G = (V, E)$ be a triconnected graph with an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$. Let $i([x, y], z)$ separate P_i. If P_i does not contain z then
i) z is an articulation point in one of the bridges of P_i, and
ii) if P_j is the largest-numbered ear that contains z, then $j > i$.

Proof Let B be the bridge of P_i containing z. Then B has an attachment in both $V_i(x, y)$ and $P_i[x, y] - \{x, y\}$, since otherwise, x, y would be a separating pair. Let a be an attachment of B in $V_i(x, y)$ and let b be an attachment of B in $P_i[x, y] - \{x, y\}$. Suppose there is a path p between a and b in B that avoids z. Then, if x, y, and z are removed from G, the vertices of $V_i(x, y)$ will remain connected to the vertices of $V_i[x, y]$ by the path p. But this is not possible since $([x, y], z)$ separates P_i. Hence, every path between a and b in B must pass through z, i.e., z is a cutpoint of B.

Let C be the connected component containing $V_i(x, y)$ in $G - \{x, y, z\}$. To prove the second claim of the lemma, we note that, by Lemma 1, P_i is the lowest numbered ear containing a vertex in C. Hence every edge (w, z) with w in C must belong to an ear numbered greater than i. By the first part of this proof, we know that there is at least one such edge (w, z). This proves the second part of the lemma.[]

Using Lemma 2, we can classify triplets separating ear P_i into two types: Type 1 separating triplets are those for which P_i contains all three vertices; type 2 separating triplets are those for which P_i contains two vertices, and the third is an articulation point in one of the bridges of P_i. Type 1 separating triplets can be further classified into three types (see figure 3): Type 1a, in which z is to the left of x and y on P_i, type 1b, in which z is to the right of x and y, and type 1c, in which z is between x and y on P_i.

a) Type 1a triplet

b) Type 1b triplet

c) Type 1c triplet

figure 3

Classification of type 1 separating triplets

Let $([x, y], z)$ be a type 2 triplet separating P_i. By Lemma 2, z is a cutpoint in a bridge, B, of P_i, and z lies on an ear P_j, $j > i$. We shall refer to such cutpoints as *high cutpoints.* Let $B_1, \cdots, B_k$ be the connected components of B - $\{z\}$, and let C be the set of remaining bridges of P_i. Then $C \bigcup_{i=1}^{k} \{B_i\}$ are the bridges of P_i in G - $\{z\}$. Let $J_i(z)$ be the ear graph of P_i in G - $\{z\}$.

Lemma 3 Let G be a triconnected graph, and let $G_i(P_i)$ be the ear graph of P_i. Then,

a) $([x, y], z)$ is a type 1 triplet separating P_i in G if and only if it is a type 1 triplet separating P_i in G_i.

b) $([x, y], z)$ is a type 2 triplet separating P_i in G if and only if (x, y) is a pair separating P_i in $J_i(z)$.

Proof We note that, since G is triconnected, every anchor bridge of P_i in G has attachments to the two endpoints of P_i, and to at least one internal vertex of P_i; we shall call this *Fact 1.* We prove parts a) and b) of the lemma separately.

a) First we note that if $([x, y], z)$ is a type 1 triplet separating P_i in the ear graph G_i then it certainly separates P_i in G.

For the reverse, two cases arise:

i) If x and y are the endpoints of P_i, then by Fact 1, $([x, y], z)$ is a type 1 triplet separating P_i if and only if every anchor bridge of P_i has exactly one internal attachment on P_i, and that attachment is at z. If this holds in G then it continues to hold in the ear graph G_i, since by coalescing such anchor bridges, we do not create any new attachments.

ii) If either x or y is not an endpoint of P_i, then no anchor bridge of P_i can have an attachment in $V_i(x, y) - \{z\}$. Once again, this condition will continue to hold if all anchor bridges are coalesced, and hence will be true in G_i if it was true in G.

b) As in case a), if (x, y) is a pair separating P_i in $J_i(z)$ then clearly $([x, y], z)$ is a type 2 triplet separating P_i in G. For the reverse, once again, two cases arise.

i) z is a high cutpoint in an anchor bridge B. Let B decompose into bridges $B_1, \cdots, B_r, C_1, \cdots, C_s$ when z is removed, where the B_j are the anchor bridges of P_i in $G - \{z\}$ and the C_k are nonanchor bridges. By Lemma 1, each B_j has all of its attachments in $P_i[x, y]$ and each C_k has all of its attachments in $P_i(x, y)$ or all of its attachments in $P_i[x, y]$. Also, since $([x, y], z)$ is a triplet separating P_i in G, any bridge of P_i other than B will have either all of its attachments in $P_i(x, y)$ or all of its attachments in $P_i[x, y]$. Hence x, y will separate P_i in $J_i(z)$.

If one of x or y is not an endpoint of P_i, then every anchor bridge B' other than B has no attachment in $V_i(x, y)$. This continues to hold in G_i as well.

ii) z is a cutpoint in a non-anchor bridge: In this case no anchor bridge of G can have an attachment in $V_i(x, y)$, and the result follows by an argument as in case i.[]

Finally, we make the following observation on the size of all of the ear graphs in G.

Observation Let G be an n-node, m-edge triconnected graph with an open ear decomposition D. Let $H_i(Q_i), i = 1, \cdots, s$ be the bridge graphs of the nontrivial ears in D, and for each i, let ear Q_i have n_i nodes, and let the bridges of Q_i in H_i have m_i edges. Then

i) $\sum_{i=1}^{s} n_i = O(n)$;

ii) $\sum_{i=1}^{s} m_i = O(n^2)$.

Proof

i) The number of nontrivial ears in G, excluding P_0, is no more than $n - 3$, and each node in G is an internal node of exactly one nontrivial ear. Hence, charging the end vertices of each nontrivial ear Q_i to its index i, we obtain $\sum_{i=1}^{s} n_i \le n + 2(n-3)$, which is $O(n)$.

ii) Each edge in G appears at most once as an internal attachment in the Q_i and at most n times as an end attachment in the Q_i, $i = 1, \cdots, s$. Hence, $\sum_{i=1}^{s} m_i \le m + 2(n-3)n$, which is $O(n^2)$.[]

Based on the characterization in Lemmas 1, 2 and 3, we obtain the following high-level algorithm to find all separating triplets in a triconnected graph.

Four Connectivity Algorithm: Finding All Separating Triplets in a Triconnected Graph $G = (V, E)$

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

2) *For* $i = r-1, r-2, \cdots, 0$ *do*

if P_i is a nontrivial ear *then*

A) Construct the ear graph $G_i(P_i)$.

B) Use $G_i(P_i)$ to find all type 1 triplets separating P_i.

C) In the bridges of P_i, find the cutpoints that lie on ears numbered higher than i, and use them to find all type 2 triplets separating P_i.

Let $|V| = n$ and $|E| = m$. Step 1 has a linear-time sequential algorithm and an $O(\log n)$ time parallel algorithm with $O(m)$ processors on a CRCW PRAM [MaScVi, MiRa]. Step 2A has a linear-time sequential algorithm and an $O(\log n)$-time parallel algorithm with $O(m \log n)$ processors on an ARBITRARY CRCW PRAM [MiRa2, RaVi].

Let n_i be the number of vertices contained in P_i, and m_i be the number of edges incident on vertices contained in P_i. In section 5.1, we present algorithms to find type 1 triplets separating a nontrivial ear P_i in $O(n_i^2 + m_i)$ sequential time, and in $O(\log n_i)$ parallel time with n_i^2 processors on an EREW PRAM. In section 5.2, we show how to find all high cutpoints in the bridges of each ear, organized in a forest of block-trees, in $\sum_i O(n + m_i)$ time plus some additional time for processing trivial ears, which is $O(m)$ over the execution of the entire algorithm. This parallelizes into an $O(\log^2 n)$ time algorithm to find cutpoints in bridges of all nontrivial ears on an ARBITRARY CRCW PRAM with n^2 processors. We use this to develop an algorithm to find all type 2 triplets in $\sum_i O(n \cdot n_i + m_i))$ sequential time, and in $O(\log^2 n)$ parallel time using $\sum_i O(n \cdot n_i + m_i)$ processors on an ARBITRARY CRCW PRAM. Thus by the Observation we obtain an $O(n^2)$ time sequential implementation of Algorithm 1, as well as an $O(\log^2 n)$ time parallel implementation on an ARBITRARY CRCW PRAM with n^2 processors.

5. Finding All Triplets that Separate an Ear

5.1. Finding Type 1 Separating Triplets

In this section we give algorithms to find type 1a, 1b and 1c separating triplets on an ear P_i. Recall that $([x, y], z)$ is a type 1 triplet separating P_i, if x, y and z lie on P_i, and the vertices in $V_i(x, y)$ are separated from the vertices in $V_i[x, y]$ when x, y and z are removed from P_i.

As shown in Lemma 3, if x and y are the endpoints of ear P_i then $([x, y], z)$ form a type 1 triplet separating P_i if and only if the anchoring star in $G_i(P_i)$ has exactly one internal attachment on P_i, and that attachment is z. This is a simple condition that can be checked in constant time with m_i processors. For finding any other type 1 triplet separating P_i, it suffices to view the ear graph $G_i(P_i)$ as the path P_i together with a collection of stars, and to identify all type 1 triplets separating P_i in G_i. For this we can work with a star graph $G(P)$ without any reference to the fact that it is the ear graph of an ear.

Let $G(P)$ be a star graph with k vertices on P, l stars, and a total of p edges on the stars. We present an $O(k^2 + p)$ time sequential algorithm and an $O(\log k)$ time parallel algorithm with $k^2 + p$ processors on an EREW PRAM to find all type 1 triplets separating P in $G(P)$. Assume that the vertices on P are numbered in order as $1, \cdots, k$ from left to right.

For a closed interval $[x, y]$ on P, let
$L[x, y]$ be the leftmost attachment among all stars that have an attachment in $[x, y]$,
$S[x, y]$ be the second leftmost attachment among all stars that have an attachment in [x,y], and
$R[x, y]$ and $M[x, y]$ be the rightmost and second rightmost attachments, respectively, of stars that have an attachment in $[x, y]$.

The following lemma is straightforward.

Lemma 4 Let x, y, z be three vertices on P. Then

a) $([x, y], z)$ is a type 1a triplet separating P if and only if $L[x+1, y-1] = z$, $S[x+1, y-1] \geq x$ and $R[x+1, y-1] \leq y$; and

b) $([x, y], z)$ is a type 1b triplet separating P if and only if $R[x+1, y-1] = z$, $M[x+1, y-1] \leq y$ and $L[x+1, y-1] \geq x$.

We compute $L[x, y]$, $S[x, y]$, $R[x, y]$ and $M[x, y]$ for every interval $[x, y]$ with $x \leq y$ by a doubling technique that first computes these values incrementally for intervals whose size is a power of 2, and then computes the values for all remaining intervals. This algorithm runs in $O(k^2 + p)$ time sequentially, and in $O(\log k)$ time on an EREW PRAM with $k^2 + p$ processors.

Algorithm 1: Finding Type 1A Triplets

1.1) *Initialize:* For $i = 1, \cdots, k$ compute $L[i, i]$, $S[i, i]$, $R[i, i]$, and $M[i, i]$. These values can be computed in $O(k + p)$ sequential time and $O(\log k)$ parallel time on an EREW PRAM with $k + p$ processors by using bucket sort to order the star edges in increasing order of attachment, with ties broken in decreasing order of the leftmost (rightmost) attachment of the star the edge belongs to for $L[i, i]$ and $S[i, i]$ (for $R[i, i]$ and $M[i, i]$).

1.2) For $j = 1, \cdots, \lceil \log k \rceil$ compute, for each i, $L[i, i + 2^j - 1]$ from $L[i, i + 2^{j-1} - 1]$ and $L[i + 2^{j-1}, i + 2^j - 1]$. Similarly compute $S[i, i + 2^j]$, $R[i, i + 2^j]$ and $M[i, i + 2^j]$. Each of these values can be computed in constant time in parallel, and hence sequentially as well. Thus, this total step takes $O(k \log k)$ time sequentially and $O(\log k)$ parallel time on an EREW PRAM with k processors.

1.3) For each pair $[x, y]$, $x < y$, let i_{xy} be the integer satisfying $x + 2^{i_{xy}} \leq y < x + 2^{i_{xy}+1}$. Compute $L[x, y]$ from the pre-computed values $L[x, x + 2^{i_{xy}} - 1]$ and $L[y - 2^{i_{xy}} + 1, y]$ in constant time. Similarly compute $S[x, y]$, $R[x, y]$ and $M[x, y]$. As in step 2, each of these values can be computed in constant time, and hence this step requires $O(k^2)$ sequential time; it is straightforward to implement this in $O(\log k)$ parallel time on an EREW PRAM with k^2 processors.

An analogous procedure identifies type 1b separating triplets.

For type 1c separating triplets, let $L[x, y]$ and $R[x, y]$ be as before. Let z_l be a vertex in $[x, y]$ which is an attachment of a star with an attachment at $L[x, y]$; analogously let z_r be a vertex in $[x, y]$ which is an attachment of a star with an attachment at $R[x, y]$. Let $S'[x, y]$ be the leftmost attachment of stars with an attachment in $[x, y]$ - $\{z_l\}$ and let $M'[x, y]$ be the rightmost attachment among stars with an attachment in $[x, y]$ - $\{z_r\}$. Then the following lemma is again straightforward.

Lemma 5 The triplet $([x, y], z)$ is a type 1c triplet separating P if and only if $S'[x+1, y-1] \geq x$, $M'[x+1, y-1] \leq y$ and one of the following three conditions hold:

a) $z_l = z_r = z$; or

b) $L[x+1, y-1] \geq x$ and $z = z_r$; or

c) $R[x+1, y-1] \leq y$ and $z = z_l$.

Using Lemma 5 we can compute the type 1c triplets separating P in a manner analogous to the method used for finding type 1a and 1b triplets separating P.

5.2. Finding Type 2 Triplets Separating an Ear

There are many implementation details in this algorithm. We give a high-level description first, and then elaborate on each of the steps. We use the result in Lemma 2 that if $([x, y], z)$ is a type 2 triplet separating P_i, then z is a high cutpoint, i.e., z is a cutpoint in one of the bridges of P_i, and z belongs to a higher-numbered ear than P_i. Observe that the number of blocks (biconnected components) and the number of articulation points in the bridges of an ear P_i is no more than n. As a matter of notation, we will denote the star(s) in the ear graph $G_i(P_i)$ corresponding to a bridge or a collection of bridges B of P_i by $s(B)$, and similarly, the bridge(s) of P_i corresponding to a star or a collection of stars S of G_i by $b(S)$. We now present the high-level algorithm for finding type 2 triplets separating P_i. For convenience we assume that the vertices of G are numbered so that any vertex contained in P_i has a smaller number than a vertex in the interior of any P_j, $j > i$.

Algorithm 2: Finding Type 2 Triplets

2.1) For each star s of G_i, we construct a list $L(s)$ of those pairs of vertices x, y on P_i for which s is the only star that has an attachment in $V_i(x, y)$ and $V_i[x, y]$. Note that there can be no more than n_i^2 entries in the lists for all of the stars of G_i, since each pair can appear on at most one list. The list for each star is in lexicographically increasing order on (x, y).

2.2) For each ear P_i, we determine the high cutpoints in each of its bridges.

2.3) For each bridge B of P_i, for each high cutpoint a in B, we find all pairs of vertices separating P_i in $P_i \bigcup (B - \{a\})$ (note that we do *not* include the remaining bridges of P_i in this graph), using the triconnectivity algorithm in [MiRa2,RaVi]. These separating pairs can be specified as candidate sets (see section 3). We maintain these candidate sets for all cutpoints for a given bridge B in a properly sorted manner; we call this the *candidate representation for* B.

2.4) We compare the entries in $L(s)$ for each s with pairs of vertices in a candidate set in the candidate representation for $b(s)$, and each match gives a type 2 separating triplet for P_i.

We need the following observation.

Observation Let z be a high cutpoint of a bridge B of P_i, and x, y, a pair of vertices on P_i. Then $([x, y], z)$ is a type 2 triplet separating P_i if and only if

a) (x, y) is a pair separating P_i in the graph $P_i \bigcup (B - \{z\})$, and
b) $s(B)$ is the only star of G_i that has an attachment in both $V_i(x, y)$ and in $V_i[x, y]$.

Proof If $([x, y], z)$ is a type 2 triplet separating P_i, then by part b of Lemma 3 we know that (x, y) separates P_i in $J_i(z)$. Hence (x, y) is certainly a pair separating P_i in $P_i \bigcup (B - \{z\})$. Further if any other bridge B' of P_i has an attachment in both $V_i(x, y)$ and $V_i[x, y]$, then removal of x, y and z leaves $V_i(x, y)$ connected with $V_i[x, y]$, which is not possible since $([x, y], z)$ separates P_i by assumption. Hence part b) of the observation must hold as well.

For the reverse, assume that parts a) and b) hold. Then it follows that x, y is a pair separating P_i in G - $\{z\}$, since by b), no bridge other than B can connect $V_i(x, y)$ with $V_i[x, y]$ in G - $\{x, y, z\}$. Hence $([x, y], z)$ must be a type 2 triplet separating P_i.[]

All pairs of vertices on P_i satisfying property b) appear on the list $L(s(B))$, which we construct in step 1. The pairs satisfying property a) are those that lie in a common candidate set in the candidate representation for B, which we construct in step 2. In step 3 we scan these two sets of pairs of vertices, and identify matches between the two sets; each such match gives a type 2 triplet separating P_i, and every type 2 triplet separating P_i appears as such a match. This establishes the correctness of the above algorithm.

We now explain how to implement steps 2.1 through 2.4 to obtain the stated time and processor bounds.

STEP 2.1

The algorithm for step 2.1 is similar to the algorithms for finding type 1 separating triplets. By Lemma 3, if x and y are the endpoints of ear P_i, then the anchoring star of G_i is the unique star containing vertices in both $V_i(x, y)$ and $V_i[x, y]$. For any other pair x, y we can work with a star graph $G(P)$ without any reference to the fact that it is the ear graph of an ear.

As in section 5.1, given a star graph $G(P)$ we compute certain values for each interval of vertices on P. The values computed are $L[x, y]$, $S''[x, y]$, $R[x, y]$ and $M''[x, y]$, where $L[x, y]$ and $R[x, y]$ are, as before, the leftmost and rightmost attachments, respectively, among all stars that have an attachment in the closed interval $[x, y]$. Let s_l be a star with attachments at $L[x, y]$ and in $[x, y]$, and similarly, let s_r be a star with attachments at $R[x, y]$ and in $[x, y]$. $S''[x, y]$ is the leftmost attachment among all stars with an attachment in $[x, y]$ except star s_l; similarly, $M''[x, y]$ is the rightmost attachment among all stars with an attachment in $[x, y]$ except s_r. From these definitions, the following lemma is straightforward.

Lemma 6 Star s is the only star that has an attachment in $V(x, y)$ and $V[x, y]$ if and only if one of the following three hold:

a) $S''[x+1, y-1] \geq x$, $R[x+1, y-1] \leq y$ and $s = s_l$; or

b) $L[x+1, y-1] \geq x$, $M''[x+1, y-1] \leq y$ and $s = s_r$; or

c) $S''[x+1, y-1] \geq x, M''[x+1, y-1] \leq y$ and $s = s_l = s_r$.

Using Lemma 6 and the method of section 5.1, we can form the lists $L(s)$ for all stars of all nontrivial ears in $O(n^2)$ sequential time and in $O(\log n)$ parallel time on an EREW PRAM with n^2 processors.

STEP 2.2

Sequential Algorithm

Let $H_i = \bigcup_{j=0}^{i} P_j$. Let $A_1, \cdots, A_k$ be the bridges of H_i. Let B_j be A_j with its attachment edges and vertices deleted. A *split* of P_i is an articulation point in one of the B_j. An *ex-node* of P_i is a vertex in one of the B_j adjacent to an attachment on H_i. An *adj-node* of P_i is an ex-node, which is adjacent to an vertex on P_i. For example, in figure 4, H_1 has four nontrivial bridges and one trivial bridge; vertices a, b and c are some of the split nodes of P_1; vertices a, d and e are some of the ex-nodes of P_1 of which a and d are adj-nodes as well. We observe that by Lemma 2, if $([x, y], z)$ is a type 2 triplet separating P_i then z is a split or ex-node of P_i or z must be an attachment of one of the A_j on H_{i-1}.

We organize the splits and ex-nodes of P_i in a forest of *split-trees* analogous to the tree of biconnected components. There is one split-tree for each B_j, whose vertices are the splits, ex-nodes and blocks of B_j. There is an edge between a split and each block it lies in, as well as an edge between each ex-node (that is not also a split) and the unique block in which it lies. For u an ex-node, let $A(u, j)$ be the jth smallest vertex adjacent to u and belonging to H_{i-1}, if it exists, *null* otherwise, for $j = 1, 2, 3, 4$. By our numbering scheme for vertices, $A(u, j), j = 1, \cdots, 4$ (when defined) represent four distinct vertices on lowest numbered ears adjacent to u. For example, in figure 4 vertex k has $A(k, 1) = 1$, $A(k, 2) = 5$, $A(k, 3) = \phi$, $A(k, 4) = \phi$. The number of entries in $A(u, j)$, over all ex-nodes u, is $O(n)$.

Let F_{i-1} be H_{i-1} with the two endpoints of P_i deleted. (In figure 4 F_0 is the single vertex 0.) Let $A(u)$ be the set of two smallest non-null vertices in $F_{i-1} \cap \{A(u, 1), A(u, 2), A(u, 3), A(u, 4)\}$. By construction, $A(u)$ contains the two smallest numbered vertices in F_{i-1} adjacent to u (when they exist), and can be obtained in constant time per ex-node, since we have the $A(u, j)$. Note that if we did *not* have the $A(u, j)$, finding the $A(u)$ would take time proportional to the number of edges incident on the ex-nodes and that could be as large as $\theta(m)$.

From the forest of split-trees we derive the forest of trees of biconnected components (or *block-trees*) of the bridges of P_i by first constructing the *augmented graph* as follows: We augment the vertex set of the forest of split-trees by adding in vertex v to represent H_{i-1}, -- a potential 'high-block' (i.e., a connected component that contains no high cutpoints), and we add in the set of vertices $U = \bigcup_{\text{ex-nodes } u} A(u)$, -- potential high cutpoints. We put in an edge between v and each vertex in U as well as edges between u and vertices in $A(u)$, for each ex-node u.

a) Graph G with open ear decomposition indicated by ear number along edges.

b) Some splits, ex-nodes and adj-nodes for P_1 in G.

figure 4
Illustrating step 2.2 for finding type 2 separating triplets

Observe that a vertex w in H_{i-1} is a high cutpoint in a bridge of P_i if and only if, for some split-tree T of P_i, w is the only vertex in F_{i-1} that is adjacent to a vertex in T. Since by construction $A(u)$ includes the two smallest vertices adjacent to u, if they lie in F_{i-1}, it follows that w is a high cutpoint in a bridge of P_i if and only if it is a cutpoint in the augmented graph. Similarly, an ex-node u in a split-tree T is a cutpoint separating vertices in T from the rest of the bridge of P_i if and only if u has an attachment in F_{i-1} and no other ex-node in T has an attachment in F_{i-1}. This again holds if and only if u is a cutpoint in the augmented graph. Hence the blocks and articulation points in this augmented graph are precisely the blocks and articulation points in the bridges of P_i. We find these in $O(n)$ sequential time, using a linear-time algorithm for biconnectivity [Ta]. At this point we have the forest of block-trees for the bridges of P_i. In additional $O(m_i)$ time, we can obtain all of the adj-nodes by scanning all edges incident on the internal vertices of P_i.

All that remains is to obtain incrementally the split-trees for P_l and the $A(u, j)$ for the new ex-nodes of P_l in an efficient way, where P_l is the next nontrivial ear. To update information for P_l, we first process the forest of split-trees for P_i to eliminate those splits and blocks that disappear and the new ones that appear when $P_i, P_{i-1}, \cdots, P_{l+1}$ are added. This is done in $O(n + m_i + l - i)$ time by finding blocks, cutpoints and ex-nodes in the graph $\bigcup_j B_j \bigcup_{k=l+1}^{i} P_k \bigcup$ {attachment edges of each B_j in the interior of P_i}. This gives us the split-trees for P_l. The new exnodes for P_l are the nodes in the interior of P_i adjacent to a vertex in H_l; in particular, this includes the nodes in the interior of P_i adjacent to its endpoints. We compute the $A(u, j)$ values for these new exnodes. This computation takes $O(m)$ time over the entire execution of the algorithm. Now we are ready to find type 2 triplets separating P_l.

Parallel Implementation of Step 2.2:

This step is similar to the algorithms in [MiRa2, RaVi] that find the ear graphs of all nontrivial ears. The only difference is that we now find the forest of block-trees instead of connected components. For this we can use any efficient parallel block finding algorithm [MaScVi, MiRa, TaVi]. By noting that the total size of the graphs present at each stage of the algorithm is $O(n^2)$, we obtain an $O(\log^2 n)$ time parallel algorithm on an ARBITRARY CRCW PRAM with n^2 processors.

STEP 2.3

Sequential Algorithm

We number the vertices in the forest of block-trees in post-order with respect to a depth first search. We label each attachment edge to P_i in the bridges of P_i by the number of the block it belongs to (since each such edge is incident on an adj-node, this is done in constant time per edge). We remove any multiple occurrences of edges with the same block number and attachment. Since the number of blocks and the number of articulation points is $O(n)$ (over all bridges of P_i) this step can be done in $O(n + m_i)$ time for all of the bridges.

We now sort (using bucket sort) the labeled attachment edges in increasing order of the attachments, with edges having the same attachment sorted in increasing order of their label, and we leave the sorted edges in stacks corresponding to their attachment number. Now, with another post-order traversal of the block-trees, we can determine, for each cutpoint s of each bridge B of ear P_i, the stars formed from B when s is deleted from B, in $O(n + m_i)$ time.

At this point, for each high cutpoint x of bridge B, we have $s(B - \{x\})$, the collection of stars formed from B when x is removed from B. Each of these stars has no more than n_i attachments. Using the algorithm in [MiRa2] we can find the separating pairs on P_i corresponding to these stars in $O(k \cdot n_i)$ time, where k is the number of stars. These are organized as the vertices on the faces of the planar embedding of the coalesced graph of $P_i \cup s(B - \{x\})$ [MiRa2]; we call this the *candidate collection* for $s(B - \{x\})$. This has an $O(k \cdot n_i)$ size representation. We find such a collection for each cutpoint. This procedure takes $O(n \cdot n_i)$ time over all cutpoints of all bridges of P_i, since the number of stars formed in all of these graphs is no more than $2n$.

In order to execute step 2.4 efficiently, we store the candidate collections in a special way. Let us confine our attention to a specific bridge B (note that the candidate collections are obtained bridge by bridge). Let X and Y be a pair of candidate sets in the set of candidate collections for B. Then we note that the spans of X and Y are either disjoint or one contains the other (the *span* of a candidate set is the interval $[a, b]$, where a is the lowest numbered and b is the highest numbered vertex in the candidate set). We represent these candidate sets in a special form called the *candidate representation of B* as follows: We maintain each candidate set as a list, a *candidate list,* with vertices ordered in increasing order of their number. We have n_i stacks, one for each vertex on P_i, and in the stack for vertex v, we place pointers to all candidate lists that contain v. These pointers are arranged in increasing order of the lowest-numbered vertex in the candidate list, with ties broken in decreasing order of the highest-numbered vertex in the candidate list (the topmost pointer points to the candidate list with the lowest numbered vertex). For each candidate list we maintain a pointer to the current lowest-numbered vertex in the candidate list; initially the pointer for each candidate list points to its lowest-numbered vertex.

Parallel Implementation of Step 2.3:

This step can be implemented on ear P_i in $O(\log^2 n)$ time with $O(n \cdot n_i)$ processors using efficient parallel algorithms for computing post-order numbering on trees [TaVi], for sorting [Co] and for finding separating pairs in a star graph [MiRa2].

STEP 2.4

Sequential Algorithm:

We scan the entries in $L(s(B))$ in order. If the current entry is (x, y), we look at the topmost candidate list R in the stack for vertex y in the candidate representation for bridge B and check its current lowest-numbered vertex z. If $z > x$ then we proceed to the next entry in $L(s(B))$. If $z = x$ then we have found a match, and hence a type 2 triplet separating P_i. If $z < x$, we move the pointer for R along the list until it points to a vertex $u \geq x$. If $u = x$, then we have located a type 2 triplet and we leave the pointer at u. If $u = y$ then we pop the pointer to R off the stack and proceed to check the next candidate list in the stack for y; if $y > u > x$ we leave the pointer at u and proceed to the next entry in $L(s(B))$. It is easy to see that this scan locates all type 2 triplets $([x, y], z)$ with z in B, and the time it takes is proportional to the sizes of $L(s(B))$ and the candidate lists for B. Hence, over all bridges of P_i this procedure takes time $O(n_i^2 + n \cdot n_i) = O(n \cdot n_i)$.

Parallel Implementation of Step 2.4:

To implement step 2.4 in parallel we allow ourselves $O(\log n)$ time per entry (x, y) on $L(s(B))$ to determine if x lies in the same candidate list as y for some entry in stack y; this is accomplished by binary search on the entries in stack y followed by a binary search on the vertices in the relevant candidate list R.

REFERENCES

[BeX] M. Becker, W. Degenhardt, J. Doenhardt, S. Hertel, G. Kaninke, W. Keber, K. Mehlhorn, S. Naher, H. Rohnert, T. Winter, "A probabilistic algorithm for vertex connectivity of graphs," *Inform. Proc. Lett.,* vol. 15, no. 3, 1982, pp. 135-136.

[Co] R. Cole, "Parallel merge sort," *Proc. 27th IEEE Ann. Symp. on Foundations of Comp. Sci.,* 1986, pp. 511-516.

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

[Ev2] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k," *SIAM J. Comput.,* vol. 4, 1975, pp. 393-396.

[EvTa] S. Even, R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM J. Comput.,* vol. 4, 1975, pp. 507-518.

[Ga] Z. Galil, "Finding the vertex connectivity of graphs," *SIAM J. Comput.,* Vol. 9, 1980, pp. 197-199.

[GiSo] M. Girkar, M. Sohoni, "On finding the vertex connectivity of graphs," *Tech. Report ACT-77,* Coordinated Science Laboratory, University of Illinois, Urbana, Il, May, 1987.

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

[Ka] A. Kanevsky, "On the number of minimum size separating vertex sets in a graph," *Tech. Report ACT-80,* Coordinated Science Laboratory, University of Illinois, Urbana, Il, July, 1987.

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

[KarRa] R. M. Karp, V. Ramachandran, "Parallel algorithms for shared memory machines," *Handbook of Theoretical Computer Science,* North Holland, 1990, to appear.

[LiLoWi] N. Linial, L. Lovasz, A. Wigderson, "A physical interpretation of graph connectivity, and its algorithmic applications," *Proc. 27th IEEE Ann. Symp. on Foundations of Comp. Sci.,* 1986, pp. 39-48.

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

[Ma] D. Matula, "Determining edge connectivity in $O(nm)$," *Proc. 28th IEEE Ann. Symp. on Foundations of Comp. Sci.,* Los Angeles, CA, 1987, pp. 252-259.

[MaScVi] Y. Maon, B. Schieber, U. Vishkin, "Parallel ear decomposition search (EDS) and st-numbering in graphs," *VLSI Algorithms and Architectures,* Lecture Notes in Computer Science vol. 227, 1986, pp. 34-45.

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

[MiRa2] G. L. Miller, V. Ramachandran, "A new graph triconnectivity algorithm and its parallelization," *Proc. 19th ACM Ann. Symp. on Theory of Computing,* New York, NY, 1987, pp. 335-344.

[RaVi] V. Ramachandran, U. Vishkin, "Efficient parallel triconnectivity in logarithmic time," *VLSI Algorithms and Architectures, AWOC 88,* Springer-Verlag LNCS 319, 1988, pp. 33-42.

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

[TaVi] R. E. Tarjan, U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM J. Comput.,* 14, 1985, pp. 862-874.

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