

- [16] V. Ramachandran. Fast and processor-efficient parallel algorithms for reducible flow graphs. Technical Report ACT-103, Coordinated Science Laboratory, University of Illinois, Urbana, IL, November 1988.
- [17] V. Ramachandran. Fast parallel algorithms for reducible flow graphs. In S. Tewksbury, B. Dickinson, and S. Schwartz, editors, *Concurrent Computations: Algorithms, Architecture and Technology*, pages 117–138. Plenum Press, New York, NY, 1988.
- [18] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In J. Reif, editor, *Synthesis of Parallel Algorithms*. Morgan-Kaufmann, New York, NY, 1992. To appear.
- [19] V. Ramachandran. Class notes. Dept. of Computer Sciences, Univ. of Texas at Austin, Spring 1990.
- [20] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. In *Proc. 3rd Aegean Workshop on Computing*, volume 319 of *Lecture Notes in Computer Science*, pages 111–123. Springer Verlag, 1988.
- [21] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.
- [22] R.E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. Comput.*, 14:862–874, 1984.
- [23] J. Westbrook and R.E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7:433–464, 1992.

- [3] D. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacements. In *Proc. ICALP 89*, volume 372 of *Lecture Notes in Computer Science*, pages 379–393, 1989. To appear in *SIAM J. Comput.* .
- [4] H.N. Gabow, J.L. Bentley, and R.E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Ann. ACM Symp. on Theory of Computing*, pages 135–143, 1984.
- [5] H.N. Gabow and R.E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. System Sci.*, 30:209–221, 1985.
- [6] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.
- [7] P. Gibbons, R.M. Karp, V. Ramachandran, D. Soroker, and R.E. Tarjan. Transitive compaction in parallel via branchings. *J. Algorithms*, 12:110–125, 1991.
- [8] X. Han. *An Algorithmic Approach to Extremal Graph Problems*. PhD thesis, Department of Computer Sciences, Princeton University, Princeton, NJ, June 1991.
- [9] X. Han, P. Kelsen, V. Ramachandran, and R.E. Tarjan. Computing minimal spanning subgraphs in linear time. In *Proc. of the Third ACM-SIAM Symp. on Discrete Algorithms*, pages 146–156, 1992.
- [10] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13:338–355, 1984.
- [11] R.M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 869–941. MIT Press/Elsevier, 1990.
- [12] J.A. La Poutré. *Dynamic Graph Algorithms and Data Structures*. PhD thesis, Department of Computer Sciences, University of Utrecht, The Netherlands, September 1991.
- [13] Y. Maon, B. Schieber, and U. Vishkin. Parallel ear decomposition search (eds) and  $st$ -numbering in graphs. *Theoret. Comput. Sci.*, 47:277–298, 1986.
- [14] G.L. Miller and V. Ramachandran. Efficient parallel ear decomposition with applications. Manuscript, January 1986.
- [15] G.L. Miller and J.H. Reif. Parallel tree contraction and its applications. In *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pages 478–489, 1985.

sequence  $L(u)$  (see preprocessing stage) of endpoints of edges of  $C$  in  $T_H$  and the preorder numbers of the children of  $u$  in  $T_H$ , this can be done in time  $O(|C| + l)$  where  $l$  is the number of children of  $u$  in  $T_H$ . To compute a minimal set  $A_2$  such that  $A_1 \cup A_2$  is good for  $G_u$ , we first collapse the vertex set of each connected component in  $G_u(A_1)$ , then collapse all vertices in the resulting graph corresponding to a component of  $G_u(A_1)$  that contains a marked vertex into a single new vertex. In the resulting graph we compute a spanning tree and let  $A_2$  be the set of those edges in  $C$  that correspond to an edge of this spanning tree. Thus, we spend  $O(p + q)$  time on all executions of step (2.3). Step (2.4) requires time proportional to the size of  $G_u$ , i.e.,  $O(|C| + l)$ . Using the techniques we described above we can implement one execution of steps (2.4.1)-(2.4.4) in time  $O(|C| + l)$ .

Altogether, we see that algorithm 12 runs in time  $O(p + q)$  as claimed.

## 6 Concluding Remarks

In this paper we have presented efficient parallel and sequential algorithms for the problems of finding a minimal 2-edge-connected spanning subgraph of a 2-edge-connected graph and finding a minimal biconnected spanning subgraph of a biconnected graph.

The algorithms for both problems have a similar high-level structure: repeatedly compute a spanning tree of the input graph with the smallest possible number of redundant edges and minimally augment this tree. This strategy is useful for finding minimal subgraphs of a graph with respect to other properties. In particular this approach gives similar algorithms for the problem of finding a minimal  $k$ -connected subgraph of a graph (for any  $k$ ), assuming we have a method for augmenting a spanning tree with respect to these properties.

In [9] we describe refinements for algorithm 1 that yield linear time sequential algorithms for the above problems. The algorithms for both problems use the linear time augmentation procedures described in section 5 as subroutines. These results reduce the parallel work required for these problems by a factor of  $\Theta(\log n)$ .

## References

- [1] P.C.P. Bhatt, K. Diks, T. Hagerup, V.C. Prasad, T. Radzik, and S. Saxena. Improved deterministic parallel integer sorting. *Inform. and Comput.*, 94:29–47, 1991.
- [2] R. Cole and U. Vishkin. Approximate and exact parallel scheduling with applications to list, tree, and graph problems. In *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, pages 478–491, 1986.

taking the union of all sets containing a vertex other than  $u$  on the path from  $v$  to  $u$  in  $T_H$  and then adding those sets having the same block number as one of these sets; we create another set by proceeding similarly for the path from  $w$  to  $u$ . If we do this for every edge of  $A$ , we get a collection of sets whose names are children of  $u$ . We determine the block number of each such set using an auxiliary graph  $G_A$ . The vertices of  $G_A$  are the children  $u_1, \dots, u_l$  of  $u$  in  $T_H$ . There is an edge in  $G_A$  from  $u_i$  to  $u_j$  if an edge of  $A$  connects a descendant of  $u_i$  with a descendant of  $u_j$ . Compute the connected components of  $G_A$ . All sets whose names are part of the same component receive the same block number, namely the smallest name in the component. We associate with each block number  $i$  the low-value  $low(i)$  of the core of the corresponding block of  $T_H$  (i.e., the minimum low-value of any vertex of  $T_H$  in the core). After we have computed  $T_c$  in step (2.5) we update the quantity  $low(i)$  as follows: let  $C_i$  be a component of  $G_A$  that gets assigned block number  $i$ . The quantity  $low(i)$  is set to the minimum low-value of any child of  $u$  in  $C_i$ . All of these steps can be performed in time  $O(|C| + l)$  where  $l$  is the number of children of  $u$  in  $T_H$ . (To construct the adjacency list for  $G_A$ , see the implementation of step (2.3) below.)

To see that the sets of  $T_H$  satisfy the two properties stated above, note that as soon as a set  $x$  becomes part of a larger set  $y$ , all sets with the same block number as  $x$  are also merged into  $y$ . Furthermore, each new set at the end of the current execution of step (2.5) has a child of  $u$  as its name. It follows that the sets of  $T_H$  do indeed satisfy the two properties mentioned above. We conclude that the total number of union and find operations is  $O(p + q)$ . Since all unions are of the form  $unite(p(z), z)$ , we can use the algorithm of Gabow and Tarjan ([5]). This gives us an overall time bound of  $O(p + q)$  for all executions of step (2.5).

We now describe the implementation of steps (2.2), (2.3), and (2.4) of algorithm 12. In step (2.2) we need to be able to check quickly whether an edge of  $C$  is incident with a leaf in  $V_l$ . For this we maintain, for each block number  $i$ , the number of children of the corresponding node in  $T_c$ . Let  $d(i)$  denote the number of children of the node in  $T_c$  corresponding to block number  $i$ . We update the  $d$ -values as follows: recall that we incorporate an edge  $e = (v, w)$  with lca  $u$  into  $T_c$  by taking the union of sets along the paths from  $v$  to  $u$  and from  $w$  to  $u$ . After we execute  $unite(z, p(z))$  where  $z$  is a vertex on one of these two paths with block number  $j$  and its parent  $p(z)$  in  $T_H$  has some block number  $k < j$ , we set  $d(k) := d(j) + d(k) - 1$ . After incorporating all the edges in  $A$  we have a collection of sets of  $T_H$  whose names are children of  $u$  in  $T_H$ . Let  $C_i$  be the connected component of  $G_A$  (see above) whose minimum name is  $i$ . As described above all sets whose name is in  $C_i$  receive block number  $i$ . We set  $d(i)$  to the sum of the  $d$ -values of the names contained in component  $C_i$ . With the  $low$ -values and  $d$ -values, we can compute the set  $A_1$  in step (2.2) in time  $O(|C|)$ .

To construct the adjacency list for  $G_u$  in step (2.3), we need to determine for each endpoint other than  $u$  of an edge of  $C$  the unique child of  $u$  of which it is a descendant in  $T_H$ . Using the sorted

$B_0$  such that  $A_1^{(2)} \cup B_2 \cup A_2$  is good for  $G_u$ . Since  $A \subseteq A_1^{(2)} \cup B_2 \cup A_2$ , the edges in  $B_2$  are essential for (P2). Finally we observe that each edge in  $A_1 - B_0$  has at least one endpoint in  $V_l$  that is not incident with another edge in  $A_1 \cup A_2$ ; it is therefore essential for (P1).  $\square$

We show how to implement the various steps so that the running time of algorithm 12 is  $O(p + q)$  (where  $p = n(H)$  and  $q = m(H)$ ).

We first describe how to maintain the block condensation  $T_c$  of  $T_H + IN$  at each step. Some methods are known (e.g., [23] and [12]) for maintaining the biconnected components of a graph under edge insertions. These methods however are superlinear in the number of edge insertions. We shall explain how  $T_c$  can be maintained in linear time.

We represent  $T_c$  by a partition of the nodes of  $T_H$ ; as before, we refer to a set in this partition as a *set of  $T_H$* . Let us first assume that each set of  $T_H$  corresponds to a node of  $T_c$ , the tree  $T_c$  being the block condensation of  $T_H + IN$ . Hence, a set of  $T_H$  is either the singleton set consisting of the root of  $T_H$  or it contains the nodes in the core of a block of  $T_H + IN$ . Below we shall see that this requirement needs to be relaxed.

In step (2.5) we compute  $T_c$  by incorporating the edges in  $A$  one by one. Fix an edge  $e = (v, w)$  of  $A$  with  $lca(e) = u$ . Adding  $e$  to  $IN$  in step (2.5) results in a block in  $T_H + IN + e$  whose root is  $u$  and whose core is the union of the cores in  $T_H + IN$  containing a vertex other than  $u$  on the fundamental cycle of  $e$  in  $T_H$ . Unfortunately, the unions of the corresponding sets of  $T_H$  are not all of the form  $unite(p(z), z)$  where  $p(z)$  denotes the parent of a node  $z$  in  $T_H$  (unless  $v = u$  or  $w = u$ ). For instance if  $u$  has two children  $v$  and  $w$  that are both in a set by themselves then the addition of edge  $e = (v, w)$  in step (2.5) amounts to taking the union of sets  $\{v\}$  and  $\{w\}$ ; this union is clearly not of the form  $unite(p(z), z)$ . Thus, the algorithm of Gabow and Tarjan ([5]) does not apply directly.

We overcome this problem by deferring some of the union operations to a later point. Inductively, we assume that the collection of sets of  $T_H$  has the following two properties: (1) the vertices in a set of  $T_H$  induce a subtree of  $T_H$ ; we take the root of that subtree as the *name* of the set. (2) The vertex set of a core of a block in  $T_H + IN$  is equal to the union of sets of  $T_H$  whose names are children of the same node of  $T_H$ . We implement this by assigning to each set a *block number*: two sets have the same block number iff they are contained in the core of a single block of  $T_H + IN$ . We choose for the block number the smallest name (w.r.t. preorder numbering) of a set of  $T_H$  that is contained in the core of a given block of  $T_H + IN$ . We maintain for each block number a list of sets that have this block number (in order to perform the union operations efficiently).

Let  $e = (v, w)$  be an edge of  $A$  with  $lca(e) = u$ . After adding edge  $e$  to  $IN$  in step (2.5), we update the sets of  $T_H$  as follows (assume  $v$  and  $w$  are both different from  $u$ ): we create one new set by

for  $G_u$ . For the only-if direction we prove the contrapositive. If a vertex of  $V_l$  is not incident with an edge of  $B$ , then this vertex represents the core of a block in  $H_0 + B$  whose root is a proper descendant of  $u$ , implying that  $H_0 + B$  is not biconnected. Also if  $B$  is not good for  $G_u$  then, in the graph  $G_u(B)$ , there exist at least two connected components at least one of which does not contain a marked vertex. The union of the cores corresponding to vertices in a component that does not contain a marked vertex forms the core of a block of  $H_0 + B$  that does not contain all the vertices of  $T_H$ , implying again that  $H_0$  is not biconnected.

Now we prove the if-part of the above claim. We consider the two cases  $u = 1$  and  $u > 1$ . If  $u = 1$  we argue as follows. Since  $H_0 + C$  is biconnected, the tree  $T_c$  has no leaf with a low-value  $> 1$  because the root of the corresponding block in  $H_0$  would be a cutpoint in  $H_0 + C$ . Since all the vertices in  $V_l$  are incident with an edge in  $B$ , it follows that there can be no cutpoint in  $H_0 + B$  other than  $u$  (the root of  $T_H$ ). Since  $B$  is good for  $G_u$ ,  $G_u(B)$  is connected and any two vertices in  $H_0 + B$  other than the root of  $T_H$  are connected by a path in  $H_0 + B$  not containing the root of  $T_H$ . Hence, the root of  $T_H$  is not a cutpoint of  $H_0 + B$  and  $H_0 + B$  is indeed biconnected. Now consider the case  $u > 1$ . As before we observe that no leaf in  $T_c$  has a low-value  $> u$ . Since any vertex in  $V_l$  is incident with an edge in  $B$ , there can be no cutpoint in  $H_0 + B$  other than  $u$ . Since  $B$  is good for  $G_u$ , there is a path between any two vertices in  $H_0 + B$  other than  $u$  that avoids vertex  $u$ . Hence,  $u$  is not a cutpoint in  $H_0 + B$ . We conclude that  $H_0 + B$  is biconnected.

We now check that  $A$  (as defined in step (2.4.5)) is a minimal subset of  $C$  such that each vertex in  $V_l$  is incident with an edge in  $A$  and  $A$  is good for  $G_u$ . (The following argument is very similar to that preceding algorithm 10.) Let us denote these two properties by (P1) and (P2), respectively. By step (2.4.5) of algorithm 12 the set  $A$  satisfies property (P2). We need to verify that each vertex in  $V_l$  is incident with an edge in  $A$  (i.e.,  $A$  satisfies property (P1)). By step (2.4.2) each vertex of  $V_l$  is incident with an edge of  $A_1^{(2)} \cup A_2$  and hence with an edge in  $A_1^{(3)} \cup A_2$ . Let  $e$  be an edge in  $B_3$ . By the definition of  $B_1$  any edge in  $B_1$  has exactly one endpoint that is incident with an edge of  $A_2$ . Since  $B_3 \subseteq B_1$ , this also holds for edge  $e$ . Again by the definition of  $B_1$  the endpoint of an edge in  $B_1$  that is not incident with an edge of  $A_2$  is incident only with this edge of  $B_1$ . It then follows from the definition of  $B_3$  that both endpoints of edge  $e \in B_3$  are incident with edges in  $A_2 \cup B_2$ . Since  $(A_2 \cup B_2) \cap B_3 = \emptyset$ , it follows that each vertex of  $V_l$  is incident with an edge in  $A$ .

Let us say that an edge  $e \in A$  is *essential for* (P1) if  $A - \{e\}$  does not satisfy property (P1) and edge  $e \in A$  is *essential for* (P2) if  $A - \{e\}$  does not satisfy property (P2). We have to show that each edge  $A$  is essential for (P1) or (P2). (Note that an edge may be essential for both properties.) By step (2.4.5) of algorithm 12 each edge in  $B_4$  is essential for (P2). Furthermore, each edge in  $B_1 - B_3$  has one endpoint in  $V_l$  that is not incident with another edge in  $A_1^{(3)} \cup A_2$ . Since  $A \subseteq A_1^{(3)} \cup A_2$ , this implies that the edges in  $B_1 - B_3$  are essential for (P1). By step (2.4.3)  $B_2$  is a minimal subset of

- (2.1) Let  $C$  be the set of nontree edges whose lca is  $u$ .  
 If  $C \neq \emptyset$ , perform steps (2.2)-(2.5) below:
- (2.2) Let  $V_l$  be the set of leaves  $v$  of  $T_c$  such that  $low(v) = u$  and the block of  $T_H + IN$  whose core is represented by  $v$  has a root different from  $u$  in  $T_H$ . Compute a minimal set  $A_1 \subseteq C$  such that each vertex in  $V_l$  is incident with an edge in  $A_1$ .
- (2.3) Define auxiliary graph  $G_u$  as follows: the vertices of  $G_u$  are the children  $u_1, \dots, u_l$  of  $u$  in  $T_H$ . To each edge  $e$  of  $C$  connecting a descendant of  $u_i$  to a descendant of  $u_j$  in  $T_H$  corresponds an edge  $(u_i, u_j)$  in  $G_u$ . Mark the vertices in  $G_u$  whose low-value is less than  $u$ . For  $B \subseteq C$  let  $G_u(B)$  be the subgraph of  $G_u$  induced by those edges in  $G_u$  that correspond to edges in  $B$ . We say that a set  $B \subseteq C$  is *good* for  $G_u$  if  $u = 1$  and  $G_u(B)$  is connected or  $u > 1$  and in  $G_u(B)$  there is a path from any vertex to some marked vertex. Compute a minimal subset  $A_2 \subseteq C$  such that  $A_1 \cup A_2$  is good for  $G_u$ .
- (2.4) Eliminate some edges from  $A_1$  as follows:
- (2.4.1) Let  $B_0$  be the subset of edges in  $A_1$  whose endpoints in  $V_l$  are incident with other edges in  $A_1 \cup A_2$ . Let  $A_1^{(1)} = A_1 - B_0$ .
- (2.4.2) Each vertex in  $V_l$  not incident with an edge in  $A_1^{(1)} \cup A_2$  selects a single edge of  $B_0$  incident with it. Let  $B_1 \subseteq B_0$  be the set of edges selected at this step and let  $A_1^{(2)} = A_1^{(1)} \cup B_1 (= (A_1 - B_0) \cup B_1)$ .
- (2.4.3) Determine a minimal subset  $B_2 \subseteq B_0$  such that  $A_1^{(2)} \cup B_2 \cup A_2$  is good for  $G_u$ . Let  $A_1^{(3)} = A_1^{(2)} \cup B_2 (= (A_1 - B_0) \cup B_1 \cup B_2)$ .
- (2.4.4) Let  $B_3$  be the set of edges in  $B_1$  whose endpoints in  $V_l$  are incident with edges in  $B_2 \cup A_2$ . Let  $A_1^{(4)} = A_1^{(3)} - B_3 (= ((A_1 - B_0) \cup B_1 \cup B_2) - B_3)$ .
- (2.4.5) Determine a minimal subset  $B_4 \subseteq B_3$  such that  $A_1^{(4)} \cup B_4 \cup A_2$  is good for  $G_u$ .  
 Let  $A = A_1^{(4)} \cup B_4 \cup A_2 (= (((A_1 - B_0) \cup B_1 \cup B_2) - B_3) \cup B_4 \cup A_2)$ .
- (2.5) Let  $IN := IN \cup A$ . Let  $T_c$  be the block condensation of  $T_H + IN$ . Update  $low$  (see below).

**Lemma 18** *Upon termination of algorithm 12,  $IN$  is a minimal augmentation for  $T_H$ .*

*Proof.* Fix an iteration of step (2). Let  $IN_u$  denote the set of edges in  $IN$  with  $lca > u$  and let  $R_u$  be the set of nontree edges with  $lca < u$ . Let  $H_0$  be the graph  $T_H + IN_u + R_u$ . As in the proof of lemma 2 it suffices to establish that  $A$  (as defined in step (2.4.4)) is a minimal subset of  $C$  such that  $H_0 + A$  is biconnected.

Assume inductively that  $H_0 + C$  is biconnected. First, we prove that for any  $B \subseteq C$  the graph  $H_0 + B$  is biconnected if and only if each vertex in  $V_l$  is incident with an edge in  $B$  and  $B$  is good

For a set named  $v$  we denote this quantity by  $d(v)$ . For some  $z$  in  $T_H$  let  $d = d(\text{find}(z))$  and  $d' = d(\text{find}(p(z)))$ . After executing  $\text{unite}(p(z), z)$  we update the  $d$ -values by setting  $d(\text{find}(z)) = d + d' - 1$ . Thus, the total time to maintain the quantities  $d(v)$  is  $O(p + q)$ . We can determine in constant time whether an endpoint  $v$  in  $T_c$  of an edge of  $C$  is a leaf of  $T_c$  by checking whether  $d(v) = 0$ . We can compute the set  $A$  in step (2.2) by eliminating the edges in  $C$  one by one, making sure not to remove all the edges incident with a vertex of  $V_l$ . By maintaining for each vertex of  $V_l$  a counter recording the number of edges of  $C$  incident with it that have not been eliminated yet, the set  $A$  can be computed in time  $O(|C|)$ .

In summary we see that algorithm 11 computes a minimal augmentation for  $T_H$  in  $H$  in time  $O(p + q)$ .

## 5.2 A Linear Time Algorithm for Computing a Minimal Augmentation for Biconnectivity

We now describe a linear time sequential algorithm for minimally augmenting a spanning tree into a biconnected graph. Let  $H$  be a biconnected graph on  $p$  vertices and  $q$  edges and let  $T_H$  be a spanning tree in  $H$ .

In a preprocessing phase we compute for each vertex in  $T_H$  the preorder number (henceforth identifying a vertex with its preorder number), we compute the lca's of nontree edges in  $H$  (using the algorithm of [10]), we sort the nontree edges by their lca, and compute for each lca  $u$  a sorted list  $L(u)$  of endpoints in  $T_H$  of nontree edges with lca  $u$  (by traversing  $T_H$  in preorder). Furthermore, we compute during a postorder traversal of  $T_H$  for each vertex  $v$  in  $T_H$  the smallest lca of a nontree edge incident with a descendant of  $v$  (we consider a vertex to be a descendant of itself). We denote this quantity by  $\text{low}(v)$ . The preprocessing phase requires time  $O(p + q)$ .

The following algorithm computes a minimal augmentation for  $T_H$  with respect to biconnectivity. As usual the set  $IN$  contains those edges that have already been chosen for the augmentation. The tree  $T_c$  represents the block condensation of  $T_H + IN$  (defined in section 4.2). For a node  $v$  in  $T_c$  define  $\text{low}(v)$  to be the minimum low-value of any node in  $T_H$  that is collapsed into  $v$ . Steps (2.2), (2.3) and (2.4) in the following algorithm are similar to algorithms 6, 7 and 10, respectively.

**Algorithm 12** Minimally augmenting  $T_H$  into a biconnected graph.

*Input* Biconnected graph  $H$ , spanning tree  $T_H$ .

*Output* Minimal subset  $IN$  of nontree edges in  $H$  such that  $T_H + IN$  is biconnected.

- (1) Initialize:  $IN := \emptyset, T_c := T_H$ .
- (2) For  $u := n$  downto 1 do:



on  $n$  vertices can be maintained under  $m$  edge insertions in time  $O(m\alpha(m, n) + n)$ . This bound is improved to  $O(m + n)$  by [12]. Although the bound of [12] matches our bound, we choose to present our method because it was discovered independently of [12]. Moreover, the presentation of our technique will be helpful in understanding the more complicated solution for the biconnected case.

The tree  $T_c$  is represented by a partition of the nodes of  $T_H$  into disjoint nonempty sets. We refer to a set in this collection as a *set of  $T_H$* . Each set of  $T_H$  has a *name* which is a vertex in this set. Initially, each vertex  $v$  of  $T_H$  is in a singleton set  $\{v\}$  whose name is  $v$ . At any point each set of  $T_H$  will represent the set of vertices of a 2-edge-connected component of  $T_H + IN$ . Hence, the vertices in this set induce a subtree in  $T_H$ . The name of the set is the root of this subtree. The following two operations access the sets of  $T_H$ :

- (i) *find*( $x$ ): returns the name of the set of  $T_H$  containing node  $x$  of  $T_H$ ;
- (ii) *unite*( $x, y$ ): merges the sets of  $T_H$  containing  $x$  and  $y$  into a new set whose name is the name of the old set containing  $x$ .

We incorporate the edges of  $A$  into  $T_c$  (step (2.3) of algorithm 11) one at a time. Let  $e = (v, w)$  be an edge in  $A$  (at the current iteration of algorithm 11) and let  $u = lca(v, w)$ . The addition of  $e$  to  $IN$  in step (2.3) creates a 2-edge-connected component in  $T_H + IN$  whose vertex set is the union of the vertex sets of the components in  $T_H + IN - e$  that contain a vertex on the fundamental cycle of  $e$  in  $T_H$ . This amounts to creating a new set of  $T_H$  that is the union of all old sets containing a node on the fundamental cycle of  $e$  in  $T_H$ . The following sequence of union and find operations creates such a set ( $p(z)$  denotes the parent of a node  $z$  in  $T_H$ ,  $e = (v, w)$  and  $u = lca(v, w)$ ):

```

 $x := find(v);$ 
while  $x \neq u$  do begin  $unite(p(x), x); x := find(p(x))$  end;
 $y := find(w);$ 
while  $y \neq u$  do begin  $unite(p(y), y); y := find(p(y))$  end;

```

All unions are of the form  $unite(p(z), z)$  for some vertex  $z$  in  $T_H$ . Hence, we may use the algorithm of Gabow and Tarjan ([5]) to perform at most  $q$  union and find operations in time  $O(p + q)$  (with  $T_H$  being the *union tree*). This is also the total time spent in step (2.3) over all iterations of the for-loop.

In step (2.2) we need to determine whether an endpoint in  $T_c$  of an edge in  $C$  is a leaf in  $T_c$ . To this end we maintain for each set of  $T_H$  the number of children of the corresponding vertex in  $T_c$ .

*Output* Minimal subset  $IN$  of nontree edges in  $H$  s.t.  $T_H + IN$  is 2-edge-connected.

- (1) Initialize:  $IN := \emptyset$  and  $T_c := T_H$ .
- (2) For  $u := n$  downto 1 do:
  - (2.1) Let  $C$  be the set of nontree edges in  $H$  with lca  $u$ . If  $C \neq \emptyset$ , perform the following two steps:
    - (2.2) Let  $V_l$  be the set of leaves of  $T_c$  such that the edge  $e$  from the leaf to its parent in  $T_c$  satisfies  $low(e) = u$ . Determine a minimal set  $A \subseteq C$  such that each vertex in  $V_l$  is incident with an edge in  $A$ .
    - (2.3) Let  $IN := IN \cup A$  and let  $T_c$  be the condensation of  $T_H + IN$ .

**Lemma 17** *Upon termination of algorithm 11,  $IN$  is a minimal augmentation for  $T_H$  in  $H$ .*

*Proof.* First, we show that  $T_H + IN$  is 2-edge-connected (upon termination of algorithm 11). We establish this by proving that every edge in  $T_H$  is covered by an edge of  $IN$ . Consider the iteration of algorithm 11 when some lca  $u$  is processed. Assume inductively that all edges of  $T_H$  with  $low$ -value  $> u$  are covered by edges in  $IN$  chosen at previous iterations. Fix an edge  $e$  from a node  $v$  to its child  $w$  in  $T_c$ . If  $low(e) = u$ , then every leaf of  $T_c$  that is a descendant of  $w$  is incident with an edge of  $C$  and the edge  $e'$  to its parent in  $T_c$  satisfies  $low(e') = u$ . Hence, all leaves of  $T_c$  that are descendants of  $w$  belong to  $V_l$  and are incident with an edge in the set  $A$  computed at this iteration. We conclude that all edges of  $T_c$  with  $low$ -value  $u$  will be covered at this iteration. Therefore, every edge of  $T_H$  is covered by an edge of  $IN$  upon termination of algorithm 11.

To see why  $IN$  is minimal, consider the same iteration of algorithm 11. Let  $e$  be an edge in  $A$  at this iteration. Since the set  $A$  constructed in step (2.2) is minimal with respect to the property that each node in  $V_l$  is incident with an edge of  $A$ , at least one endpoint of  $e$  in  $V_l$ , say  $y$ , is incident with no other edge of  $A$ . Moreover, the edge  $e'$  from  $y$  to its parent in  $T_c$  is covered neither by an edge of  $IN$  whose lca is  $> u$  (by the definition of  $T_c$ ) nor by an edge with lca  $< u$  (since  $low(e') = u$ ). Hence,  $e'$  is covered only by  $e$ . It follows that all the edges in  $A$  are essential in the final graph  $T_H + IN$ .  $\square$

We show how to implement algorithm 11 so that it runs in time  $O(p + q)$  (where  $p = n(H)$  and  $q = m(H)$ ).

The tree  $T_c$  is the condensation of  $T_H + IN$ . Hence, each vertex in  $T_c$  represents the set of vertices of a 2-edge-connected component of  $T_H + IN$ . We shall describe a method for maintaining  $T_c$  that runs in time  $O(p + q)$ . Our technique is similar to methods discovered independently by [23] and [12]. One result in [23] shows that the 2-edge-connected components of an initially connected graph

steps (0), (1) and (1.1) of algorithm 1 can be implemented to execute in linear time by making use of linear time procedures for finding an ear decomposition and for vertex triconnectivity given in [18] (see section 3.3). Thus, if we adhere to the high-level structure of algorithm 1, we obtain algorithms for computing a minimal 2-edge-connected or biconnected spanning subgraph that run in time  $O(m + n \log n)$ . Recently ([9]), linear time algorithms have been developed for these problems. These algorithms use the linear time augmentation procedures described in this section as subroutines. Similar linear time augmentation procedures (as well as linear time algorithms for finding minimal spanning subgraphs) have been found independently by [8].

### 5.1 A Linear Time Algorithm for Computing a Minimal Augmentation for 2-Edge-Connectivity

We shall first describe how to minimally augment a spanning tree with respect to 2-edge-connectivity. Assume we are given a 2-edge-connected graph  $H$  on  $p$  vertices and  $q$  edges and a spanning tree  $T_H$  in  $H$  rooted at an arbitrary vertex. We describe how to compute a minimal augmentation for  $T_H$  in  $H$ , i.e., a minimal set of nontree edges of  $H$  whose addition to  $T_H$  yields a 2-edge-connected graph. The sequential algorithm for finding a minimal augmentation for  $T_H$  is simpler than the parallel algorithm because it processes in one step a single lca instead of a collection of leaf chains.

In a preprocessing phase we number the vertices of  $T_H$  in preorder from 1 to  $p$ . Henceforth, we shall identify a vertex with its preorder number. We compute the lca's of nontree edges in  $H$  in linear time using the algorithm of [10]. We sort the nontree edges in  $H$  by their lca (in increasing order), and we compute for each edge  $e = (x, y)$  of  $T_H$  the quantity  $low(e)$  (or  $low(x, y)$ ) defined as follows:  $low(e)$  is the smallest lca of a nontree edge in  $H$  that covers  $e$ . Let  $(u, v)$  be an edge in  $T_H$  from a vertex  $u$  to its child  $v$ . The function  $low$  satisfies the recurrence:

$$low(u, v) = \min(\{low(v, w) : w \text{ child of } v\} \cup \{lca(e) : e \text{ is a nontree edge incident with } v\}).$$

Thus, we may compute the  $low$ -value for each edge of  $T_H$  in a postorder traversal of  $T_H$ . Altogether, the preprocessing requires time  $O(p + q)$  (where  $p = n(H)$  and  $q = m(H)$ ).

The following algorithm computes a minimal augmentation for  $T_H$ . Its structure is similar to that of algorithm 2: the main difference is that we process a single lca in each stage rather than a collection of lca's. As in algorithm 2 the set  $IN$  denotes the set of nontree edges that have been committed to the augmentation. The tree  $T_c$  represents the condensation of  $T_H + IN$  (defined in section 3.2), i.e.,  $T_c$  is obtained by collapsing the 2-edge-connected components in  $T_H + IN$ .

**Algorithm 11** Minimally augmenting  $T_H$  into a 2-edge-connected graph.

*Input* 2-edge-connected graph  $H$ , spanning tree  $T_H$  in  $H$ .

**Algorithm 10** Pruning  $A_1$ .

*Input* Graph  $H'_3$ , edge set  $A_1$ .

*Output* Biconnected spanning subgraph  $H''_3$  of  $H'_3$  such that each edge of  $A_1$  contained in  $H''_3$  is essential.

- (1) Let  $B_0$  be the set of edges of  $A_1$  whose special endpoints in  $V_l$  are incident with edges of  $A'_2 \cup A_3$ . Let  $H^{(1)} = H'_3 - B_0$ .
- (2) Each exposed vertex of  $V_l$  selects a single edge of  $B_0$  incident with it. Let  $B_1 \subseteq B_0$  be the set of edges selected at this step and let  $H^{(2)} = H^{(1)} + B_1 (= H'_3 - B_0 + B_1)$ .
- (3) Compute a minimal augmentation  $B_2 \subseteq B_0$  for  $H^{(2)}$  using algorithms 7, 8 and 9. Let  $H^{(3)} = H^{(2)} + B_2 (= H'_3 - B_0 + B_1 + B_2)$ .
- (4) Let  $B_3$  be the set of edges in  $B_1$  whose endpoints in  $V_l$  are incident with edges of  $A'_2 \cup A_3 \cup B_2$ . Let  $H^{(4)} = H^{(3)} - B_3 (= H'_3 - B_0 + B_1 + B_2 - B_3)$ .
- (5) Compute a minimal augmentation  $B_4$  for  $H^{(4)}$  using algorithms 7, 8 and 9.
- (6) Let  $H''_3 = H^{(4)} + B_4 (= H'_3 - B_0 + B_1 + B_2 - B_3 + B_4)$ .

Let  $A'_1$  be the set of edges in  $A_1$  that are contained in  $H''_3$ . From the discussion preceding algorithm 10 we get the following result.

**Lemma 16**  $H''_3$  is biconnected and every edge in  $A'_1$  is essential in  $H''_3$ .

**Corollary 4** The set  $A'_1 \cup A'_2 \cup A_3$  forms a minimal augmentation for  $H_0 = T_H + IN + R$ .

*Proof.* By lemma 16 the graph  $H''_3$  is biconnected and the edges of  $A'_1$  are essential in  $H''_3$ . By lemma 13 the edges of  $A_3$  are essential in  $H_3$ . Since  $H''_3$  is a biconnected spanning subgraph of  $H_3$ , the edges of  $A_3$  are also essential in  $H''_3$ . Finally, by lemma 15 the edges of  $A'_2$  are essential in  $H'_3$ . Since  $H''_3$  is a biconnected spanning subgraph of  $H'_3$ , the edges of  $A'_2$  are also essential in  $H''_3$ .  $\square$

An analysis similar to the one done in section 3.3 shows that the algorithm for computing a minimal biconnected spanning subgraph runs within the same resource bounds as the algorithm for finding a minimal 2-edge-connected spanning subgraph.

## 5 Sequential Algorithms

In this section we give linear time algorithms for minimally augmenting a spanning tree into a 2-edge-connected graph or into a biconnected graph. Both for 2-edge-connectivity and biconnectivity

Our goal is to remove from  $H'_3$  a subset of edges in  $A_1$  such that the resulting graph is biconnected and all edges of  $A_1$  in this graph are essential. Let  $V_l$ , as defined in algorithm 6, denote the set of leaves in the block condensation of  $H_0 = T_H + IN + R$  that represent the cores of blocks whose root does not lie on a stem in  $T_H$ . Recall that the set  $A_1$  forms a minimal subset in  $C$  with the property that each vertex in  $V_l$  is incident with at least one edge of  $A_1$ . Thus, each edge in  $A_1$  has at least one endpoint in  $V_l$  that is not incident with another edge of  $A_1$ . We call such an endpoint *special*.

By lemma 9 an edge  $e \in A_1$  can only be redundant in  $H'_3$  if each of its special endpoints (it has at least one) is incident with an edge of  $A'_2 \cup A_3$ . Let  $B_0 \subseteq A_1$  contain all those edges in  $A_1$  whose special endpoints are incident with an edge of  $A'_2 \cup A_3$ . All edges of  $A_1$  that will be removed from  $H'_3$  in stage 5 will belong to  $B_0$ .

We start by removing the edges in  $B_0$  from  $H'_3$ . Let  $H^{(1)}$  denote the resulting graph (i.e.,  $H^{(1)} = H'_3 - B_0$ ). Note that a vertex  $v$  in  $V_l$  may not be incident with an edge of  $C$  in  $H^{(1)}$ . In this case we say that vertex  $v \in V_l$  is *exposed* in  $H^{(1)}$ . We take care of this problem by having each exposed vertex in  $V_l$  select a single edge of  $B_0$  incident with it. Denote the set of selected edges by  $B_1$  (a subset of  $B_0$ ) and let  $H^{(2)}$  denote the graph  $H'_3 - B_0 + B_1$ . Although no vertex of  $V_l$  is exposed in  $H^{(2)}$  (i.e., every vertex in  $V_l$  is incident with an edge of  $C$  in  $H^{(2)}$ ), the graph  $H^{(2)}$  may not be biconnected. Since each vertex of  $V_l$  is incident with an edge of  $C$  in  $H^{(2)}$ , we may use algorithms 7, 8 and 9 to compute a minimal augmentation  $B_2$  for  $H^{(2)}$ . Let  $H^{(3)}$  denote the graph  $H^{(2)} + B_2 (= H'_3 - B_0 + B_1 + B_2)$ . By lemmas 13 and 15 the graph  $H^{(3)}$  is indeed biconnected and all edges of  $B_2$  are essential in  $H^{(3)}$ .

We are not done yet since some edges of  $B_1 \subseteq A_1$  may be redundant in  $H^{(3)}$ . Note that an edge  $e \in B_1$  can only be redundant in  $H^{(3)}$  if both of its endpoints in  $V_l$  are incident with some edge in  $A'_2 \cup A_3 \cup B_2$ . Let  $B_3$  be the subset of those edges in  $B_1$  that have this property. Remove the edges in  $B_3$  from  $H^{(3)}$  and call the resulting graph  $H^{(4)}$  ( $= H'_3 - B_0 + B_1 + B_2 - B_3$ ). Note that no vertex of  $V_l$  is exposed in  $H^{(4)}$ . Thus, we can compute a minimal augmentation  $B_4 \subseteq B_3$  for  $H^{(4)}$  using algorithms 7, 8 and 9. Let  $H''_3$  denote the resulting graph, i.e.,  $H''_3 = H^{(4)} + B_4 (= H'_3 - B_0 + B_1 + B_2 - B_3 + B_4)$ . Again by lemmas 13 and 15 the graph  $H''_3$  is indeed biconnected and the edges of  $B_4$  are essential in  $H''_3$ . We have to show that all edges of  $A_1$  that belong to  $H''_3$  are essential in  $H''_3$ . We observed earlier that the edges of  $B_2$  are essential in  $H^{(3)} = H'_3 - B_0 + B_1 + B_2$ . Since  $H''_3$  is a biconnected spanning subgraph of  $H^{(3)}$ , the edges of  $B_2$  are also essential in  $H''_3$ . Finally, we note that the edges in  $B_1 - B_3$  all have an endpoint in  $V_l$  that is not incident with an edge of  $A'_2 \cup A_3 \cup B_2 \cup B_4$  and not incident with another edge of  $B_1$ . Therefore, they are essential in  $H''_3$ . We conclude that all edges of  $A_1$  are essential in  $H''_3$ .

The following is a more compact description of the algorithm that we have just outlined.

(2.2) Determine if  $H^*$  has a cutpoint that belongs to  $B$ . If this is the case, put the edge back into the graph  $H^*$ .

(3) Let  $H'_3 = H^*$ .

Denote by  $A'_2$  the subset of edges in  $A_2$  that are contained in  $H'_3$ .

**Lemma 15**  $H'_3$  is a spanning biconnected subgraph of  $H$  and each edge of  $A'_2$  is essential in  $H'_3$ .

*Proof.* We first observe a more general fact. Let  $H'$  be a subgraph of  $H$  containing the tree  $T_H$ . Let  $e$  be a nontree edge in  $H'$  and let  $w$  be a vertex in  $H'$  not contained on the fundamental cycle of  $e$  in  $T_H$ . We claim that  $w$  is a cutpoint in  $H'$  if and only if  $w$  is a cutpoint in  $H' - e$ . The *only if* part is clear. The *if* part of the statement follows from the observation that any path in  $H'$  avoiding  $w$  yields such a path in  $H' - e$ : simply replace each occurrence of  $e$  on this path by the path in  $T_H$  connecting the endpoints of  $e$ .

We put this observation to use as follows: fix an execution of steps (2.1) and (2.2). Let  $H'$  and  $H''$  be the graph  $H^*$  before and after this execution. Let  $H'$  be biconnected. Suppose that after the execution of step (2.1) a vertex of  $B$  becomes a cutpoint of  $H^*$ . By the previous observation we must have removed an edge  $e$  in step (2.1) whose fundamental cycle contains  $w$ . Since all edges removed in step (2.1) belong to nonadjacent blocks of  $H_2$ , their fundamental cycles in  $T_H$  are vertex-disjoint. Hence, in step (2.1) we removed a *unique* edge  $e$  whose fundamental cycle includes  $w$ . Note that  $e$  is put back in step (2.2). By the above observation  $w$  is a cutpoint of  $H''$  if and only if  $w$  is a cutpoint in  $H'$ . Since  $H'$  is biconnected,  $w$  is not a cutpoint of  $H''$ . Because this holds for any vertex  $w$ , the graph  $H''$  is biconnected. Applying the above observation one more time we see that the graph  $H' - e$  has  $w$  as a cutpoint. Thus,  $e$  is essential in  $H'$ . Since  $H''$  is a biconnected spanning subgraph of  $H'$ ,  $e$  is also essential in  $H''$ . By lemma 11 each edge of  $A_2$  that is redundant in  $H_3$  will be considered in some execution of step (2.1). The claim follows.  $\square$

Above we have shown how to minimally augment graph  $H_1$  into the biconnected graph  $H'_3$  using algorithms 7, 8 and 9. The minimal augmentation is based on the fact that graph  $H_1$  has all of its cutpoints on stems of leaf chains in  $T_H$ . Any spanning subgraph of  $H$  containing  $T_H$  and having this property can be minimally augmented using algorithms 7, 8, and 9.

In the fifth and final stage of the minimal augmentation procedure we remove a subset of edges of  $A_1$  that are redundant in  $H'_3$ . This stage is done roughly as follows: remove from  $H'_3$  a maximal set of edges in  $A_1$  such that the resulting subgraph has all of its cutpoints on stems. Minimally augment this subgraph into a biconnected graph using algorithms 7, 8, and 9. Repeat these two steps one more time. We now give a more detailed description of this stage and establish its correctness.

lemma 11 only the edges in  $A_2^{(1)}$  may possibly be removed. The idea is to remove from  $H_3$  several edges in  $A_2^{(1)}$  at once and check if the blocks of  $H_2$  that contain them now contain a cutpoint. We shall put back only those edges for which the corresponding block does indeed contain a cutpoint. This strategy will work provided that the fundamental cycles in  $T_H$  of the edges are vertex disjoint. To guarantee this, we shall select at each step a subset of edges that do not lie on adjacent blocks of  $H_2$  (i.e., blocks that intersect the same stem and share a common vertex).

The following result (reminiscent of observations 1 and 2) tells us that we only need to look at at most 4 edges of  $A_2^{(1)}$  in each block of  $H_2$ .

**Lemma 14** *At most 4 vertices of any block of  $H_2$  are incident with an edge of  $A_3^L$ .*

*Proof.* Fix a vertex  $B_j$  on  $P_L$ . By the definition of  $C_L$  (see algorithm 8, step (2)), an edge of  $A_3^L$  can only be incident in  $H_2$  with a vertex in block  $B_j$  of  $H_2$  if the corresponding edge of  $C_L$  has its upper endpoint (i.e., the one closer to  $B_0$  on  $P_L$ ) at  $B_{j+1}$  or its lower endpoint at  $B_{j-1}$  or is incident with  $B_j$ . By observation 2 at most 2 edges of  $A_L$  are incident with  $B_j$ . By observation 1 at most one edge of  $A_L$  has its upper endpoint at  $B_{j+1}$  and at most one edge of  $A_L$  has its lower endpoint at  $B_{j-1}$ . We conclude that at most 4 edges of  $A_3^L$  are incident with vertices in  $B_j$ . Since the two endpoints of an edge of  $A_3$  lie in different blocks of  $H_2$ , the claim follows.  $\square$

**Algorithm 9** Removing redundant edges from  $A_2$ .

*Input* Biconnected graph  $H_3$ , edge set  $A_2$ .

*Output* Biconnected spanning subgraph  $H_3'$  of  $H_3$  such that every edge of  $A_2$  contained in  $H_3'$  is essential in  $H_3'$ .

(0) Let  $H^* = H_3$ .

Process the blocks of  $H_2$  in two phases: in phase 1 we perform steps (1) and (2) in parallel for each block  $B$  of  $H_2$  that is at an even distance from  $B_0$  on some  $P_L$ ; in phase 2 we perform these steps in parallel for each block  $B$  that is at an odd distance from  $B_0$  on some  $P_L$  (for all leaf chains  $L$ ). (This ensures that in each phase we consider a collection of edges of  $A_2^{(1)}$  whose fundamental cycles are vertex-disjoint.)

(1) Determine the subset  $V_B$  of vertices in the block  $B$  of  $H_2$  that are incident with an edge of  $A_3^L$ . (By lemma 14 we have  $|V_B| \leq 4$ .)

(2) Process the vertices  $v$  of  $V_B$  sequentially as follows:

(2.1) If  $v$  is a node in the core of an external block of  $H_1$  incident with a unique edge of  $A_2^{(1)}$ , remove that edge from the graph  $H^*$ .

blocks of  $H_2$  are also blocks of  $H_2 + e$ .  $\square$

We reduce the problem of computing a minimal augmentation for  $H_2$  to the problem of covering stem edges considered in the previous section (algorithm 4).

**Algorithm 8** Augmenting  $H_2$  into a biconnected graph.

*Input* Graph  $H_2$ , edge set  $C - A_1 - A_2$ .

*Output* Minimal subset  $A_3$  of  $C - A_1 - A_2$  such that  $H_2 + A_3$  is biconnected.

In parallel for each leaf chain  $L$ , do :

- (1) Let  $P_L$  be the path  $B_0 \dots B_k$ , i.e., the vertices of  $P_L$  are the blocks  $B_0 \dots B_k$  and there is an edge from  $B_{i-1}$  to  $B_i$  for  $i = 1, \dots, k$ .
- (2) If several edges in  $C - A_1 - A_2$  (whose lca's lie on leaf chain  $L$ ) have the same range, eliminate all but one edge having this range (for any range  $(i, j)$ ). Denote the resulting subset of  $C - A_1 - A_2$  by  $C_0$ . By mapping each edge in  $C_0$  of range  $(i, j)$  to the edge  $(B_i, B_j)$ , we obtain a set  $C_L$  of edges connecting vertices of  $P_L$ . Compute a minimal subset  $A_L \subseteq C_L$  whose edges cover the edges on  $P_L$  using the same method as in algorithm 4. Let  $A_3^L \subseteq C_0$  be the collection of those edges in  $C_0$  that are mapped to an edge in  $A_L$ .

Let  $A_3 = \bigcup_L A_3^L$ .

Let  $H_3$  be the graph  $H_2 + A_3$ .

**Lemma 13** *The set  $A_3$  is a minimal augmentation for  $H_2$ , i.e.,  $H_3$  is biconnected and all edges of  $A_3$  are essential in  $H_3$ .*

*Proof.* With lemma 12 it follows that  $H_3$  is biconnected if and only if each edge of  $P_L$  is covered by an edge of  $A_L$  (i.e., it lies on the fundamental cycle in  $P_L$  of an edge of  $A_L$ ) for any  $L$ . Since  $H_2 + C$  is biconnected, the edges in  $C_L$  cover the edges in  $P_L$  (for each  $L$ ). By lemma 6 algorithm 4 produces a minimal subset  $A_L$  of  $C_L$  covering the edges of  $P_L$ . We conclude that  $\bigcup_L A^L$  is a minimal set of edges covering all the  $P_L$ 's and hence  $A_3$  is a minimal augmentation for  $H_2$ .  $\square$

As in the last section we are now faced with the problem that edges chosen at earlier stages may have become redundant because of edges added during later stages. The procedure for pruning excess edges is more complicated here than in the last section because we have added edges in three different stages.

In stage 4 of our minimal augmentation procedure we eliminate redundant edges in  $A_2$  by resorting to a strategy similar to that used in the last section for removing edges of  $A_1$  (algorithm 5). By



edge in  $(H_2 - e) + F$  connects a vertex in  $V_0$  with a vertex outside  $V_0$  other than  $s$ . Suppose for a contradiction that there exists a nontree edge  $e'$  with this property. No edge in  $H_2 - e$  has this property. Thus,  $e'$  belongs to  $F$ . Note that if  $e'$  has an endpoint in the core of a special block, then  $e'$  is not an external edge and the other endpoint of  $e'$  is contained in an external block rooted at  $s$  (the root of the special blocks). By the definition of  $V_0$  this external block is contained in  $B_1$ . This contradicts the assumption that  $e'$  is essential in  $H_2 + F$ . Hence,  $e'$  does not have an endpoint in the core of a special block. It follows that the fundamental cycle of  $e'$  contains edges from both  $B_1$  and  $B_2$ . But then  $e_F$  is redundant in  $H_2 + F$ , contradicting the minimality of  $F$ .

We now prove the second part of lemma 11. Suppose that a critical endpoint  $v \in V_e$  of an edge  $e \in A_2^{(1)}$  is not incident with an edge in  $F$ . Consider the spanning subgraph of  $Q$  induced by the edges in  $A_2^{(2)}$ . Let  $U$  be the unique connected component in this subgraph that contains  $v$ . Note that no vertex in  $U$  is incident with an external edge in  $A_2^{(1)} - e$ . By assumption no edge of  $F$  is incident with  $v$ . It follows that no external edge in  $(H_2 - e) + F$  is incident with a vertex in  $U$ . Thus we can use the argument for part 1 of this proof to show that the root of the blocks whose cores are represented by vertices in  $U$  is a cutpoint in  $(H_2 - e) + F$ . Hence  $e$  is essential in  $H_2 + F$ .  $\square$

If  $T$  is reduced to a single node, every block of  $H_1$  is an external block having as root node the root of  $T_H$ . Thus  $V_e = \emptyset$ ,  $A_2^{(1)} = \emptyset$  and  $H_2 = H_1 + A_2^{(2)}$  is biconnected because  $H_1 + C$  is biconnected. In this case, by lemma 11 (with  $F = \emptyset$ ), all edges in  $A_2$  are essential in  $H_2$ . Thus, we may immediately proceed to algorithm 10 which removes redundant edges in  $A_1$ . Henceforth, we shall assume that  $T$  is not reduced to a single node. Thus, there is a unique block, say  $B_0$ , that has the root of  $T_H$  as its root node. We denote the root of  $B_0$  by  $r_0$ .

We shall now describe stage 3 of our minimal augmentation procedure. In this stage we compute a minimal augmentation for  $H_2$ , i.e., a minimal set  $A_3 \subseteq C$  such that  $H_2 + A_3$  is biconnected. For the following discussion we fix a leaf chain  $L$  of  $T$ . Number the vertices on the stem of  $L$  in  $T_H$  that are roots of blocks of  $H_2$  consecutively as  $r_1 \dots r_k$ , starting at the root closest to  $r_0$ . By lemma 8 and lemma 10 no two distinct blocks in  $H_2$  have the same root. Denote the block of  $H_2$  having root  $r_j$  by  $B_j$ .

Define the *range* of an edge  $e = (u, v)$  of  $C$  whose lca lies on  $L$  as the integer pair  $(i, j)$ ,  $i \leq j$ , such that  $u$  is a vertex in  $B_i$  but is different from  $r_{i+1}$  and  $v$  is a vertex in the core of  $B_j$ . Note that the fundamental cycle of an edge with range  $(i, j)$  intersects exactly the blocks  $B_i \dots B_j$  in at least one edge. As an immediate consequence of this observation we get the following result.

**Lemma 12** *Let  $e$  be an edge of  $C$  whose lca lies on leaf chain  $L$  and whose range is  $(i, j)$  ( $i \leq j$ ). Then the blocks  $B_i \dots B_j$  of  $H_2$  are all contained in a single block of  $H_2 + e$  while the remaining*

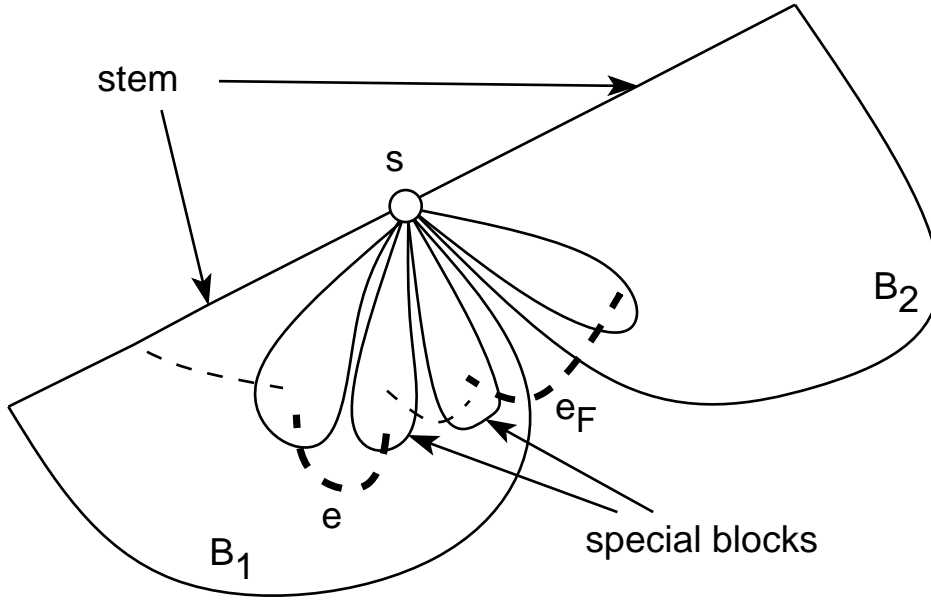


Figure 9:  $B_1$  and  $B_2$  are internal blocks of  $H_2$ . The two special blocks (rooted at  $s$ ) are blocks of  $H_1$  contained in  $B_1$ . Edge  $e$  belongs to  $A_2^{(2)}$  and edge  $e_F$  is contained in  $F$ .

connected component *special*. Let  $s$  denote the common root in  $T_H$  of all special blocks. In  $H_2 - e$  the root  $s$  of the special blocks is a cutpoint since any path from a vertex in a special block to a vertex outside the special blocks passes through  $s$ . Thus, if  $F = \emptyset$ , our claim is proved. Henceforth, assume  $F \neq \emptyset$ .

If no edge of  $F$  is incident with a vertex in a core of a *special block*, then  $s$  is a cutpoint in  $(H_2 - e) + F$  and hence  $e$  is essential in  $H_2 + F$ . Thus, suppose that some edge  $e_F \in F$  is incident on a vertex in a special block (see figure 9). By the definition of a special block no external edge is incident with the core of a special block. It follows that  $e_F$  is not external, i.e., it links a vertex in a special block with a vertex in some other external block of  $H_1$  having root  $s$ . Since each of these two blocks is contained in a single internal block of  $H_2$ , the fundamental cycle of  $e_F$  in  $T_H$  contains edges of at most two internal blocks of  $H_2$ . Thus, because  $e_F$  is essential in  $H_2 + F$  (by the assumption of the lemma), it contains edges of exactly two internal blocks of  $H_2$ , say  $B_1$  and  $B_2$ . Both of these blocks are internal in  $H_2$ . Hence, one of them, say  $B_1$ , is rooted at  $s$  and the other block is rooted at a proper ancestor of  $s$  in  $T_H$ . This is illustrated in figure 9.

Without loss of generality all the special blocks are contained in  $B_1$  (the case where they are contained in  $B_2$  is treated similarly). Thus, edge  $e_F$  connects a vertex in the core of a special block with a vertex in  $B_2$  (see figure 9). Let  $V_0$  be the set of vertices other than  $s$  that belong to  $B_1$  but do not belong to a special block or that belong to a block rooted at a proper descendant of  $s$  in  $T_H$ . We shall show that  $s$  is a cutpoint in  $(H_2 - e) + F$  by establishing that no nontree

Case 1:  $v_B \in V_e$ , i.e.,  $v_B$  is incident with an external edge in the set  $A_2^{(1)}$  computed in step (2) of algorithm 7. Let  $w$  be the endpoint of  $e$  in  $H_1$  that does not belong to an external block in  $H_1$  having  $r_B$  as its root. If the fundamental cycle of  $e$  in  $T_H$  contains an edge of a stem,  $B$  will be contained in an internal block of  $H_2$ . Otherwise the fundamental cycle of  $e$  in  $T_H$  intersects a stem in  $T_H$  only in  $r_B$ . By the definition of an external edge this is only possible if  $w$  belongs to an internal block  $B'$  with root  $r_B$ . Since  $w \neq r_B$ , the edge in  $T_H$  from  $w$  to its parent belongs to  $B'$  and also lies on the fundamental cycle of  $e$ . Again,  $B$  will be contained in an internal block of  $H_2$ .

Case 2:  $v_B \in V_l - V_e$ , i.e.,  $v_B$  is not incident with an external edge. First, we claim that there is a path in  $Q$  from  $v_B$  to a vertex in  $V_e$  (and hence  $V_e \neq \emptyset$ ). Assume for a contradiction that there is no such path. Then there exists a connected component in  $Q$  that does not contain a vertex in  $V_e$ . The nodes in this component represent the cores of external blocks sharing the same root node  $r_B$  (since none of them is incident with an external edge). Furthermore, the union of these cores forms the core of a block in  $H_2 + C$  rooted at  $r_B$ . If  $T$  is not reduced to a single node, there is at least one more block in  $H_2 + C$  (whose root is the root of  $T_H$ ), contradicting the biconnectivity of  $H_2 + C$ . Thus, in the subgraph of  $Q$  induced by the edges in  $A_2^{(2)}$  there is a simple path from  $v_B$  to a vertex  $v \in V_e$  such that none of the internal vertices on this path belongs to  $V_e$ . This path yields a simple path in  $H_2$ , not intersecting a stem, from a vertex in the core of  $B$  to a vertex in the core of a block  $B'$  rooted at  $r_B$  with the property that some vertex in the core of  $B'$  is incident with an (external) edge in  $A_2^{(1)}$ . Hence,  $B$  and  $B'$  are contained in a larger block  $B''$  of  $H_2$ . From the analysis for case 1 we know that  $B''$  is an internal block of  $H_2$ .  $\square$

In the third stage of the minimal augmentation procedure we shall compute a minimal augmentation  $A_3$  for  $H_2$ . The following result says something about the interplay between the edges of  $A_2$  and those of  $A_3$ . It will be used in the proof of lemma 15. A *critical endpoint* of an edge of  $A_2^{(1)}$  is an endpoint of that edge in  $V_e$  that is not incident with another edge of  $A_2^{(1)}$ . Note that each edge of  $A_2^{(1)}$  has at least one critical endpoint. Recall that an edge of  $C$  is *external* if it links a vertex in the core of some external block  $B$  of  $H_1$  with a vertex that lies outside of all the external blocks of  $H_1$  that have the same root as  $B$ .

**Lemma 11** *Let  $F \subseteq C$  be a minimal augmentation for  $H_2$ . All edges of  $A_2^{(2)}$  are essential in  $H_2 + F$ . Moreover, the critical endpoints of an edge in  $A_2^{(1)}$  that is redundant in  $H_2 + F$  are incident with edges of  $F$ .*

*Proof.* First, we show that an arbitrary edge  $e \in A_2^{(2)}$  is essential in  $H_2 + F$ . Consider the spanning subgraph of  $Q$  induced by the edges in  $A_2^{(2)}$ . By the minimality of  $A_2^{(2)}$  the removal of  $e$  from this subgraph yields a connected component in the subgraph that does not contain a vertex of  $V_e$  and hence no core of a block in  $H_1$  represented by a vertex in this component is incident with an external edge in  $A_1^{(1)}$ . Let us call a block of  $H_1$  whose core is represented by a vertex in this

all blocks in  $H_1$  are internal, then the removal of a cutpoint  $v$  will break up the graph in exactly two components: one component contains all vertices other than  $v$  that lie in blocks whose root is  $v$  or a vertex below  $v$  on the stem containing  $v$  while the other component contains all remaining vertices (other than  $v$ ). In this case augmenting  $H_1$  into a biconnected graph amounts to finding a set of edges connecting these two components for any cutpoint  $v$  – a relatively easy task as we shall see later. The following algorithm removes external blocks.

**Algorithm 7** Eliminating external blocks in  $H_1$ .

*Input* Graph  $H_1$ , edge set  $C - A_1$ .

*Output* Minimal set  $A_2 \subseteq C - A_1$  such that all the blocks of  $H_1 + A_2$  are internal.

- (1) Compute the block condensation of  $H_1$ . Let  $V_l$  denote the set of leaves in the block condensation representing the cores of external blocks of  $H_1$ .
- (2) Call an edge of  $C - A_1$  *external* if it links a vertex in the core of some external block  $B$  of  $H_1$  with a vertex that lies outside of all the external blocks of  $H_1$  that have the same root as  $B$ . Let  $V_e$  be the subset of nodes in  $V_l$  incident with an external edge. Using the method of algorithm 3 compute a set  $A_2^{(1)}$  of *external* edges in  $C - A_1$  that is minimal with respect to the property that each vertex in  $V_e$  is incident with at least one edge in  $A_2^{(1)}$ .

- (3) Consider the graph  $Q$  whose vertices are the vertices in  $V_l$  and whose edges are the edges of  $C - A_1$  between them.

If  $V_e = \emptyset$ , compute a minimal set  $A_2^{(2)} \subseteq C - A_1$  such that the subgraph of  $Q$  induced by the edges in  $A_2^{(2)}$  is connected.

If  $V_e \neq \emptyset$ , compute a minimal set of edges  $A_2^{(2)} \subseteq C - A_1$  such that each vertex in  $Q$  is connected to some vertex in  $V_e$  through a path in  $Q$  consisting of edges in  $A_2^{(2)}$ . We compute such a set of edges by collapsing the nodes of  $V_e$  into a single vertex and letting  $A_2^{(2)}$  be the set of edges in a spanning tree of the resulting graph.

Let  $A_2 = A_2^{(1)} \cup A_2^{(2)}$ .

Let  $H_2$  be the graph  $H_1 + A_2$ . Recall that  $T$  represents  $T_{lca}$  at the current stage of tree contraction.

**Lemma 10** *If  $T$  is not reduced to a single node, then all the blocks of  $H_2$  are internal.*

*Proof.* Let  $B$  be an external block of  $H_1$  and let  $v_B$  denote the vertex of  $V_l$  representing the core of  $B$  in the block condensation of  $H_1$ . Let  $r_B$  denote the root of block  $B$  in  $T_H$ . By the definition of an external block  $r_B$  belongs to a stem of a leaf chain. We shall show that  $B$  is contained in an internal block of  $H_2$ . We consider 2 cases.

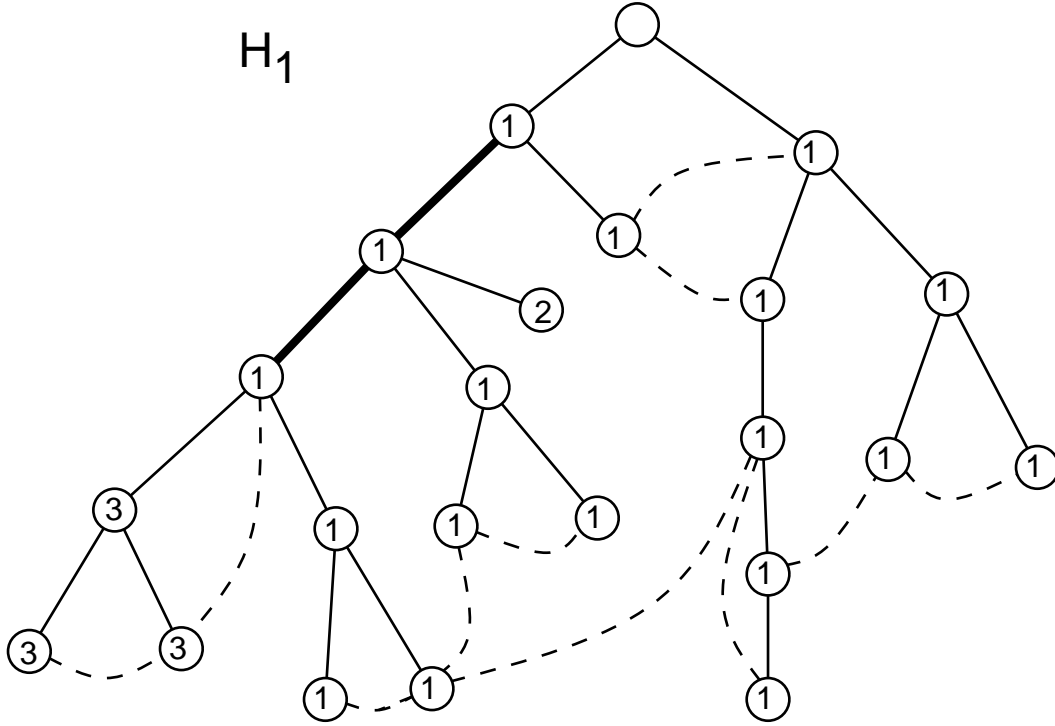


Figure 8: Graph  $H_1 = T_H + IN + R + A_1$ . Stem edges are indicated by thick solid lines. The graph contains three blocks whose core vertices are numbered 1, 2, 3. The core vertices numbered 1 belong to an internal block of  $H_1$ ; the two other blocks are external.

Conversely, if some vertex in  $V_l$  is not incident with an edge in  $F$ , then this vertex represents the core of a block in  $H_0 + F$  whose root is a cutpoint in  $H_0 + F$  that does not lie on a stem in  $T_H$ .  $\square$

The following result is immediate with lemma 9.

**Corollary 3** *The set  $A_1$  is minimal with the property that all cutpoints of  $H_0 + A_1$  lie on stems in  $T_H$ .  $\square$*

Let  $H_1$  denote the graph  $H_0 + A_1$  ( $=T_H + IN + R + A_1$ ). We classify the blocks of  $H_1$  into 2 classes. An *external block* of  $H_1$  is a block whose root lies on some stem in  $T_H$  but does not include any edge of this stem; a block is *internal* if it is not external. Figure 8 shows the graph  $H_1 = T_0 + IN + R + A_1$  for the example in figure 7. The graph  $H_1$  in figure 8 consists of 3 blocks whose core vertices are numbered 1, 2, and 3. The block whose core vertices are numbered 1 is the only internal block; it is rooted at the root of  $T_H$ . The other two blocks are external.

In stage 2 of our minimal augmentation procedure we construct a minimal set of edges  $A_2 \subseteq C - A_1$  such that  $H_1 + A_2$  has no external blocks. We eliminate external blocks for the following reason: we want to find a (minimal) set of edges whose addition to  $H_1$  will yield a biconnected graph. If

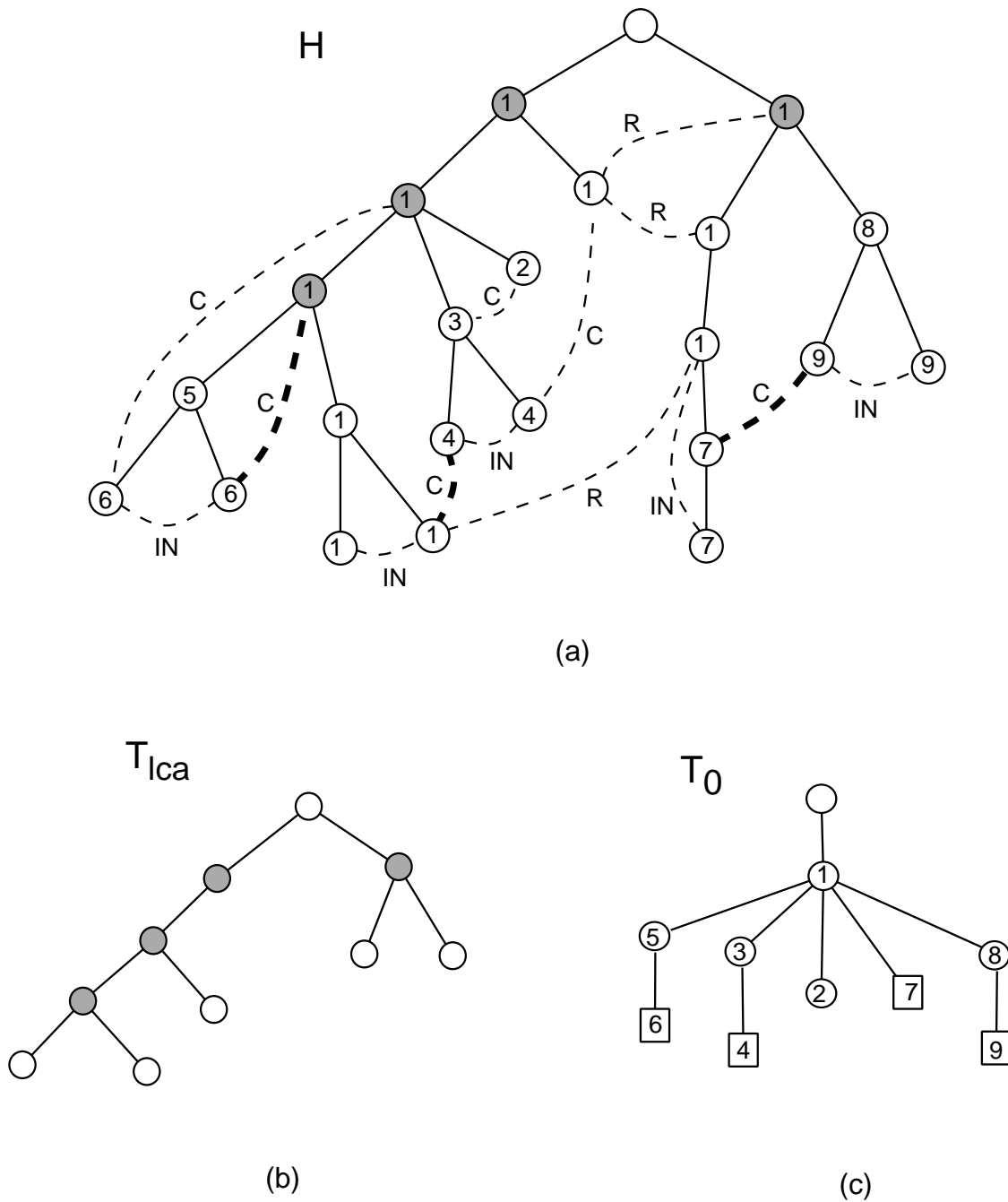


Figure 7: (a) Graph  $H$  with spanning tree  $T_H$  indicated by solid lines. Thick dashed lines represent edges of  $C$  put into  $A_1$ . The gray vertices are the lca's of nontree edges in  $C$ . Vertices with the same number belong to the core of the same block of  $T_H + IN + R$ . (b) Tree  $T_{lca}$ . Gray vertices represent those lca's currently processed. (c) Block condensation  $T_0$  of  $T_H + IN + R$ . The square vertices are the leaves in  $V_l$ . Vertex  $i$  represents the core of a block of  $T_H + IN + R$  whose vertices are numbered  $i$  in (a) ( $i = 1 \dots 9$ ).

of tree contraction and the path in  $T_H$  corresponding to leaf chain  $L$  of  $T$  is called the *stem of  $L$  (in  $T_H$ )*. We shall compute a minimal augmentation for  $H_0$  in five stages. At each of the first three stages we shall add a set of edges to achieve a certain property. In the final two “cleanup stages” we shall get rid of excess edges to obtain a minimal augmentation. Overall our method is somewhat similar to that used for 2-edge-connectivity but it is more complicated.

An example is given in figure 7. Figure 7(a) shows the graph  $H$  with the tree  $T_H$ ; the corresponding tree  $T_{lca}$  is depicted in figure 7(b). In this example we consider the second iteration of algorithm 2: the lca’s of nontree edges processed at this iteration are represented by the gray vertices in figure 7(a) and 7(b). Note that the gray vertices are exactly the vertices on the stems of leaf chains in  $T_H$ . The edges in  $IN$ ,  $R$  and  $C$  are accordingly marked in figure 7(a).

In the first stage we select a minimal set of edges  $A_1 \subseteq C$  whose addition to  $H_0$  results in a graph all of whose cutpoints lie on stems of leaf chains. The following algorithm accomplishes this. It makes use of algorithm 3.

**Algorithm 6** Eliminating cutpoints that do not lie on a stem.

*Input* Graph  $H_0 = T_H + IN + R$ , edge set  $C$ .

*Output* Minimal subset  $A_1$  of  $C$  such that  $H_0 + A_1$  has all its cutpoints on stems in  $T_H$ .

- (1) Compute the block condensation  $T_0$  of  $H_0$  (see figure 7(c)).
- (2) Let  $V_l$  be the set of leaves of  $T_0$  that correspond to the cores of blocks whose root (in  $T_H$ ) is not a vertex on a stem (in  $T_H$ ). Using the method of algorithm 3 compute a minimal set  $A_1 \subseteq C$  such that each leaf in  $V_l$  (i.e., the core collapsed into this leaf) is incident with at least one edge in  $A_1$ .

In figure 7(c) the square vertices represent the leaves in  $V_l$ . The thick dashed lines in figure 7(a) represent a minimal set of nontree edges in  $C$  such that each leaf in  $V_l$  (i.e., a vertex in the corresponding core) is incident with an edge in this set.

**Lemma 9** *For any  $F \subseteq C$ ,  $H_0 + F$  has all its cutpoints on stems iff each vertex in  $V_l$  is incident with at least one edge in  $F$ .*

*Proof.* Assume that each vertex in  $V_l$  is incident with an edge in  $F$ . Consider a block  $B$  in  $H_0$  whose root  $v$  is a cutpoint of  $H_0$  that does not lie on a stem of  $T_H$ . All leaves of  $T_0$  (block condensation of  $H_0$ ) that are descendants (in  $T_0$ ) of the vertex in  $T_0$  corresponding to the core of  $B$  belong to  $V_l$  since their roots do not lie on a stem of a leaf chain. Thus these leaves are incident with edges of  $F$ . The fundamental cycles of these edges in  $F$  contain  $v$  and a vertex on the stem of some leaf chain. Since this holds for any block of  $H_0$  whose root is  $v$ , vertex  $v$  is not a cutpoint in  $H_0 + F$ .

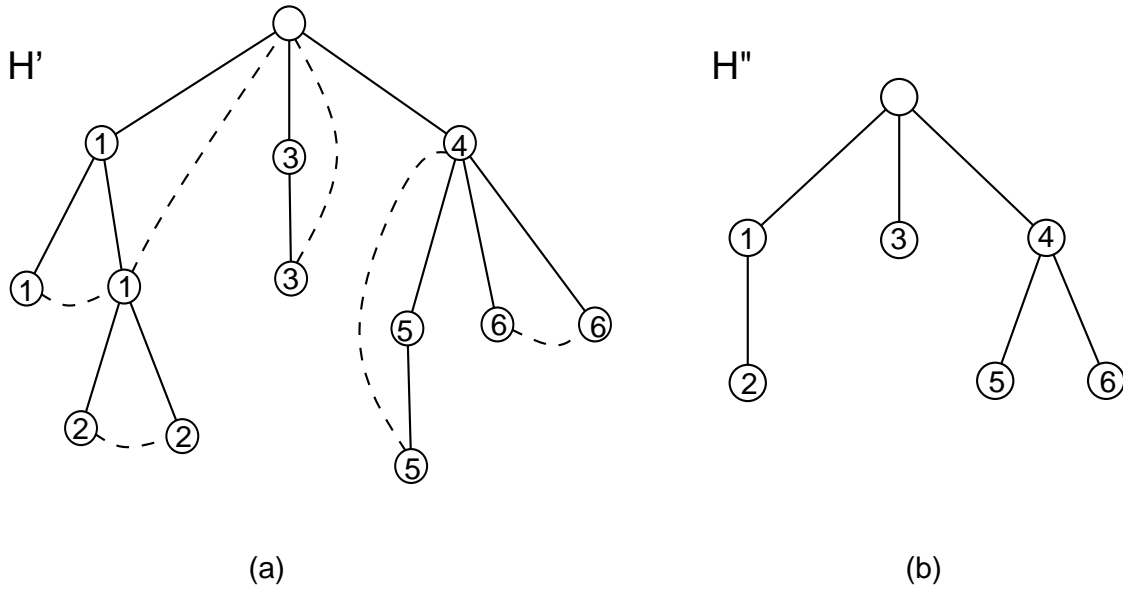


Figure 6: (a) The graph  $H'$ ; the edges in  $T_H$  are indicated by solid lines. Vertices numbered  $i$  belong to the core of the block collapsed into the vertex numbered  $i$  in  $H''$ . (b) The block condensation  $H''$  of  $H'$ .

Let  $H'$  be a subgraph of  $H$  containing tree  $T_H$  and let  $B$  be a block of  $H'$ . By lemma 8 the intersection of  $B$  with  $T_H$  is a subtree of  $T_H$ . We call the root of that subtree (i.e., the vertex closest to the root of  $T_H$ ) the *root of  $B$* . We call the subset of vertices of  $B$  that are different from the root of  $B$  the *core of  $B$* . As an easy consequence of lemma 8 we see that the cores of two distinct blocks of  $H'$  have an empty intersection. The *block condensation* of  $H'$  is obtained by collapsing the core of each block of  $H'$  into a single vertex (see section 2 for the definition of collapsing) and replacing in the resulting multigraph multiple edges by single edges. Note that the block condensation of  $H'$  is a tree. We root the tree at the root of  $T_H$ . A simple example of a graph and its block condensation is given in figure 6.

There is a natural correspondence between an edge  $e$  of  $H$  that does not belong to  $H'$  and the edge  $e'$  linking those vertices in the block condensation of  $H'$  into which the endpoints of  $e$  in  $H'$  have been collapsed. We shall usually not differentiate between an edge connecting vertices of  $H'$  and the corresponding edge connecting vertices in the block condensation of  $H'$ . For instance, we shall say that an edge of  $H$  is *incident* with a vertex  $v$  in the block condensation of  $H'$  if the corresponding edge connecting vertices in the block condensation is incident with this vertex.

We shall now describe how to compute a minimal augmentation  $A \subseteq C$  for  $H_0 = T_H + IN + R$  (step (2.2) of algorithm 2), where  $C$  denotes the set of nontree edges examined at the current iteration of algorithm 2. As in the previous section,  $T$  denotes the tree representing  $T_{lca}$  at the current stage



As for 2-edge-connectivity we remark that an alternative (sequential) method of Han and Tarjan ([8]) for computing a minimal biconnected spanning subgraph can be parallelized; the parallel implementation is similar to our method but it avoids the explicit computation of redundant and essential edges.

The proof of lemma 1 carries over to show that there is a spanning tree in  $H$  that contains at most  $2/3$  of the redundant edges in  $H$ . Hence,  $O(\log n)$  iterations of the while-loop of algorithm 1 (suitably modified) will yield a minimal biconnected spanning subgraph of  $G$ .

## 4.2 Computing a Minimal Augmentation in Parallel

We shall now describe a parallel algorithm for finding a minimal augmentation for  $T_H$  (with respect to biconnectivity). Again, the high-level structure is the same as that given in the previous section (algorithm 2; replace “2-edge-connected” by “biconnected”). The proof of lemma 2 carries over to show that the modified version of algorithm 2 does indeed compute a minimal augmentation for  $T_H$  (with respect to biconnectivity).

Recall that during the execution of the algorithm the set of nontree edges is partitioned into disjoint subsets  $IN$ ,  $C$ , and  $R$ :  $IN$  contains the edges that have already been committed to the minimal augmentation,  $C$  is the set of edges examined during this iteration of tree contraction (i.e., whose lca lies on a leaf chain of  $T$ ), and  $R$  contains those edges that are to be considered at future iterations.

Consider one iteration of the while-loop of algorithm 2. Let  $H_0$  denote the graph  $T_H + IN + R$  after step (2.1) of this iteration. We are left with the problem of finding a minimal augmentation for  $H_0$ , i.e., a minimal subset of edges in  $C$  such that its addition to  $H_0$  will result in a biconnected graph (step (2.2) of algorithm 2). We shall define an operation on graphs called *block condensation*. It is reminiscent of the condensation of a graph defined in the last section. The concept of a block condensation will be helpful in explaining various steps of our minimal augmentation procedure.

We need the following result.

**Lemma 8** *Let  $H'$  be a subgraph of  $H$  containing tree  $T_H$ . The intersection of a block of  $H'$  with  $T_H$  forms a tree.*

*Proof.* It suffices to show that any two vertices  $v, w$  in a block  $B$  of  $H'$  are connected by a path containing only tree edges (edges of  $T_H$ ) that belong to  $B$ . Since  $B$  is a connected subgraph of  $H'$ ,  $v$  and  $w$  are connected by some path  $P$  in  $B$ , possibly containing nontree edges. Note that if a nontree edge belongs to  $B$  then all the tree edges on its fundamental cycle in  $T_H$  belong to  $B$  as well. Hence we may replace on path  $P$  each nontree edge by a path of tree edges in  $B$ , obtaining a path in  $B$  consisting only of tree edges.  $\square$

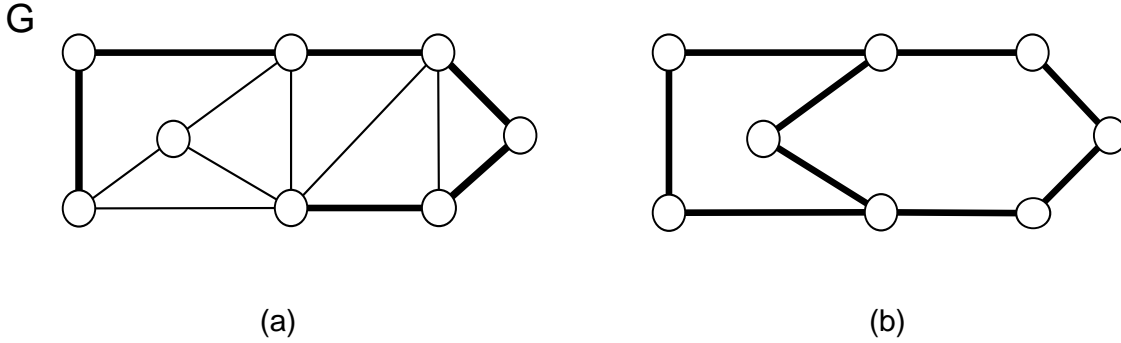


Figure 5: (a) A biconnected graph  $G$  and (b) a minimal biconnected spanning subgraph of  $G$ . Essential and redundant edges are indicated by thick and thin lines, respectively.

and will require new techniques. Figure 5 shows a biconnected graph  $G$  and a minimal biconnected spanning subgraph of  $G$ .

#### 4.1 The High-Level Algorithm

As in the previous section we start out by defining the notions of redundant and essential edges. Given a biconnected graph  $H$ , we say that an edge  $e$  of  $H$  is *redundant in  $H$*  if  $H - e$  is biconnected; an edge of  $H$  is *essential in  $H$*  if it is not redundant in  $H$ . As in the last section we shall sometimes omit  $H$  if it is clear from the context. In figure 5 essential and redundant edges are indicated by thick and thin lines, respectively. Since the graph in figure 5(b) is a minimal biconnected spanning subgraph of the graph in figure 5(a), all of its edges are essential.

We use the high-level strategy given by algorithm 1 (replace “2-edge-connected” by “biconnected”). We compute a biconnected spanning subgraph  $H$  of  $G$  with fewer than  $2n$  edges by finding an open ear decomposition for  $G$  ([3], [18]) and removing all trivial ears. In one iteration of algorithm 1 we do the following: first, we find a spanning tree  $T_H$  in  $H$  (the current biconnected spanning subgraph of  $G$ ) that includes a minimum number of redundant edges in  $H$ ; next, we determine a *minimal augmentation* for  $T_H$ , i.e., a minimal set  $B$  of nontree edges in  $H$  such that  $T_H + B$  is biconnected; finally, we update  $H$  to be the graph  $T_H + B$ .

To identify the redundant edges in  $H$  at each iteration, we construct the graph  $H'$  from  $H$  by adding a new vertex  $v_e$  for each edge  $e$  in  $H$  and replacing  $e$  by two edges  $(u, v_e)$  and  $(v_e, v)$  where  $e = (u, v)$ . A *separating pair (of vertices) in  $H'$*  is a pair of vertices in  $H'$  whose removal disconnects  $H'$ . We note that an edge  $e$  of  $H$  is redundant if and only if it does not occur in any separating pair of  $H'$  of the form  $\{v_e, u\}$  for some vertex  $u$  of  $H$ . An efficient parallel algorithm for finding all separating pairs in  $H'$  ([3], [18]) can thus be modified to identify all redundant edges efficiently.

answer maximum queries whose endpoints lie in the same segment of  $A$ , each processor constructs in time  $O(\log n)$  a *Cartesian* tree for its segment (see [4]) and preprocesses the tree in  $O(\log n)$  time so that *lca* queries can be answered in constant time (using the algorithm of [10]). As shown in [4] each maximum query whose endpoints lie in the same segment is answered by computing the *lca* of the endpoints in the Cartesian tree for this segment; this takes constant time. Altogether we see that we can answer each maximum query in constant time after a preprocessing phase that requires  $O(\log n)$  time with  $n/\log n$  processors.

To compute a maximal path in  $D_L$  (step (3) of algorithm 4), transform  $D_L$  into an undirected forest rooted at the sinks in  $D_L$ . We compute the adjacency lists for this forest  $B$ -optimally using bucket sort. We can then compute the maximal path in  $D_L$  optimally in time  $O(\log n)$  by applying the Euler tour technique ([22]) to each tree in the forest. We conclude that one execution of stage 2 can be done  $B$ -optimally.

In stage 3 (algorithm 5) we compute the cutedges in  $H^*$  by determining the 2-edge-connected components of  $H^*$ . This can be done  $A$ -optimally. Hence algorithm 5 can be implemented  $A$ -optimally.

In summary we see that one iteration of the while-loop of algorithm 2 runs in time  $O(\log n)$  with  $A(n, 2n)$  ARBITRARY processors (or in time  $O(\log^2 n)$  with  $A(n, 2n)$  EREW processors). Since algorithm 2 terminates after  $O(\log n)$  iterations (stages of tree contraction) and algorithm 1 makes  $O(\log n)$  calls to algorithm 2, algorithm 1 runs in time  $O(\log^3 n)$  with  $C(n, m)/\log^2 n + A(n, 2n)$  processors on an ARBITRARY PRAM (or  $O(\log^4 n)$  time with  $C(n, m)/\log^2 n + A(n, 2n)$  processors on an EREW PRAM). Thus, the *work* done by our algorithms (time-processor product) is  $C(n, m) \log n + A(n, 2n) \log^3 n$  on an ARBITRARY PRAM. In [9] a linear time sequential algorithm for computing a minimal 2-edge-connected spanning subgraph is given. That algorithm can be parallelized. The parallel version uses our parallel minimal augmentation procedure as a subroutine. The work done by this modified algorithm is  $C(n, m) \log n + A(n, 2n) \log^2 n$  on ARBITRARY (i.e., the work is roughly improved by a factor of  $\log n$ ). The same improvement applies to the related problem of computing a minimal biconnected spanning subgraph (see next section).

## 4 Finding a Minimal Biconnected Spanning Subgraph in Parallel

The problem considered in this section is: given a biconnected graph  $G$ , find a minimal biconnected spanning subgraph of  $G$ , i.e., a biconnected spanning subgraph of  $G$  that does not have a biconnected spanning subgraph of  $G$  as a proper subgraph. There is an obvious similarity between this problem and the one discussed in the last section. Indeed, some techniques used in the last section will be applicable here. Several new problems, however, will arise in the context of biconnectivity

sibling in  $T_{lca}$ . Note that  $w$  has a sibling in  $T_{lca}$  iff the vertex  $z$  at position  $n_v + 1 + nd(w)$  in  $A$  satisfies  $d(z) = d(w) (= d(v) + 1)$ . We conclude that the asymptotic complexity of computing the leaf chains in  $T_{lca}$  is the same as that of applying the Euler tour technique to  $T_H$ , i.e.,  $O(\log n)$  time with  $n/\log n$  processors ([22]).

We shall now examine the complexity of one iteration of the while-loop in algorithm 2. In one such iteration we shall need to compute connected or 2-edge-connected components in various graphs. Note that we can compute (2-edge-)connected components  $C$ -optimally provided we have the adjacency lists for the graphs. Since many of the graphs are obtained by collapsing subsets of vertices, it is not clear that we can obtain the adjacency list for the graphs without resorting to sorting. For that reason we shall only claim that we can compute (2-edge-)connected components  $A$ -optimally (instead of  $C$ -optimally). (This will not affect the overall complexity of one iteration of the while-loop in algorithm 2 since we require sorting in algorithm 4.)

To prepare for stage 1 (algorithm 3) we compute the condensation of  $T_H + IN + R$ . This amounts to computing the 2-edge-connected components of  $T_H + IN + R$ , which can be done using an ear decomposition algorithm ([14], [13], [18]) that is  $A$ -optimal. For step (1) assume that each vertex in  $V_l$  is assigned a unique number in the range  $1 \dots n$ . By making each vertex of  $V_l$  choose a single edge of  $C$  to a lower numbered vertex in  $V_l$  (if there is such an edge) we obtain a spanning forest in  $G_l$ . We compute the adjacency lists for this forest  $B$ -optimally. For step (2.1) we apply the Euler tour technique to each tree in the forest. All remaining steps of algorithm 3 take time  $O(\log n)$  with  $n/\log n$  processors. Thus, stage 1 can be done  $A$ -optimally.

We now come to stage 2 (algorithm 4). We compute  $T_1$  (condensation of  $T_0 + A_1$ ) by determining the 2-edge-connected components in  $T_0 + A_1$ ; this can be done  $C$ -optimally. The complexity of the remaining steps is dominated by the construction of  $D_L$  in step (2). We can reduce the problem of determining the edges of  $D_L$  to the following problem: given a sequence of  $\leq n$  numbers determine the maxima for  $\leq 2n$  (possibly overlapping) intervals of this sequence. This can be done in time  $O(\log n)$  on  $n/\log n$  processors as follows. Suppose a sequence of  $n$  numbers is given in an array  $A[0 \dots n - 1]$ . Processor  $i$  is assigned the *segment*  $A[(i - 1) \cdot \log n, \dots, i \cdot \log n - 1]$ . It computes the maximum of its segment as well as the maxima of subintervals of its segment of the form  $A[(i - 1) \cdot \log n, \dots, r]$  or  $A[s, \dots, i \cdot \log n - 1]$  where  $(i - 1) \cdot \log n \leq r, s \leq i \cdot \log n - 1$ . Note that we can compute these maxima in time  $O(\log n)$  with  $n/\log n$  processors. We store the  $n/\log n$  maxima of the segments in a second array  $B[0 \dots n/\log n - 1]$ . For simplicity we assume that  $n/\log n$  is of the form  $2^k$  for some integer  $k$ . We compute the maxima of all intervals in  $B$  of the form  $B[j \cdot 2^i, (j + 1) \cdot 2^i - 1]$  for  $0 \leq i \leq k$  and  $0 \leq j < \frac{n}{2^i \log n}$ . The total number of such intervals is  $O(n)$ . It is straightforward to compute the maxima of these intervals in time  $O(\log n)$  with  $n/\log n$  processors. With the above maxima we can answer in constant time each maximum query of the form  $\max\{A[i], A[i + 1], \dots, A[j - 1], A[j]\}$  where  $i$  and  $j$  belong to different segments. In order to

integers in the range  $0 \dots n^{O(1)}$  in time  $O(\log n)$ ;  $C(n, m)$  denotes the number of processors needed to compute connected components of a graph with  $n$  nodes and  $m$  edges represented by adjacency lists in time  $O(\log n)$ ; finally,  $A(n, m)$  stands for  $\max\{B(n), C(n, m)\}$ . As mentioned above we have currently  $B(n) = n \log \log n / \log n$  (on ARBITRARY) and  $C(n, m) = (n + m)\alpha(m, n) / \log n$  (on ARBITRARY). A step in our algorithm is *A-optimal* if it runs in time  $O(\log n)$  with  $A(n, 2n)$  processors and is *C-optimal* if it runs in time  $O(\log n)$  with  $C(n, 2n)$  processors. In these definitions we have replaced  $m$  by  $2n$  since step (0) of algorithm 1 reduces the number of edges that need to be processed to less than  $2n$ .

In the following analysis the time and processor bounds that do not specify a particular PRAM model all hold for the ARBITRARY CRCW PRAM model. We assume that the input graph  $G$  is represented by its adjacency lists. Step (0) of algorithm 1 is executed only once. Its complexity is dominated by that for finding an ear decomposition in  $G$ . This can be done in time  $O(\log n)$  using  $C(n, m)$  processors ([14, 13, 18]). As shown earlier (corollary 1),  $O(\log n)$  iterations of algorithm 1 yield a minimal 2-edge-connected spanning subgraph of  $G$ . We analyze the work done in one such iteration.

We identify redundant edges in  $H$  by finding separating pairs of edges in  $H$ . For this we modify the algorithm for finding triconnected components given in [3, 18] (see also [19]). Thus, we can compute separating pairs of edges *A-optimally*. The complexity of computing  $T_H$  is the same as that of computing connected components on a graph with  $n$  nodes and at most  $2n$  edges; we thus compute  $T_H$  *C-optimally*.

Algorithm 2 finds a minimal augmentation for  $T_H$  using  $O(\log n)$  levels of tree contraction. We prepare algorithm 2 by computing the lca's of nontree edges in  $H$  and identifying the vertices in  $T_H$  that are lca's of nontree edges; this can be done optimally in time  $O(\log n)$  using the algorithm of [20]. Next we construct the tree  $T_{lca}$ . We shall not compute an explicit representation of the tree (in terms of adjacency lists) but rather compute enough information to identify the leaf chains quickly. Using the Euler tour technique ([22]) on tree  $T_H$  we compute the following three quantities for each lca  $v$  in  $T_H$ : the number of lca's in  $T_H$  that precede  $v$  in preorder; the number of lca's on the unique path from the root of  $T_H$  to  $v$ ; the number of descendants of  $v$  in  $T_H$  (including  $v$ ) that are lca's of nontree edges. We denote these three quantities by  $num(v)$ ,  $d(v)$  and  $nd(v)$ , respectively. We call an lca  $v$  that has at least two children in  $T_{lca}$  a *split vertex*. Note that an lca  $v$  (other than the root of  $T_H$ ) belongs to a leaf chain of  $T_{lca}$  iff none of its descendants in  $T_H$  is a split vertex. Once we have determined the split vertices in  $T_H$ , we can check the latter property using one more application of the Euler tour technique (on tree  $T_H$ ). We compute the split vertices with the help of the three quantities  $num(v)$ ,  $d(v)$  and  $nd(v)$  as follows: using the quantities  $num(v)$  we write the lca's in  $T_H$  in preorder into an auxiliary array  $A$ . Note that a vertex  $v$  at position  $n_v = num(v)$  in  $A$  is a split vertex iff the vertex  $w$  at position  $n_v + 1$  in  $A$  satisfies  $d(w) = d(v) + 1$  and it has a

follows that all edges of  $T_0$  are covered by edges of  $A'_1 \cup A_2$  and therefore  $T_0 + A'_1 + A_2$  is indeed 2-edge-connected.

We now show that each edge of  $A'_1$  is essential in  $T_0 + A'_1 + A_2$ . Fix  $e \in A_1$ . A *critical endpoint* of  $e \in A_1$  is an endpoint of  $e$  in  $T_0$  that is a leaf in  $T_0$  and that is not incident with another edge of  $A_1$ . By lemma 4 and lemma 5 each edge in  $A_1$  has at least one critical endpoint in  $T_0$ . Edge  $e$  can only be redundant in  $T_0 + A_1 + A_2$  if its critical endpoints are incident with edges of  $A_2$ . In that case  $e$  will be removed in step (1.1.2.1). It is put back only if its removal produces a cutedge in  $H^*$  among the edges of  $T_0$  covered by it. Since different edges removed at one time in step (1.2.1) cover disjoint sets of edges of  $T_0$ ,  $e$  is put back only if it is indeed essential at this point. Since we do not add edges to  $H^*$  (the current subgraph of  $T_0 + A_1 + A_2$ ) after this point, an edge that is put back in step (1.1.2.2) is essential in  $T_0 + A'_1 + A_2$ .  $\square$

**Theorem 1** *The set  $A = A'_1 \cup A_2$  is a minimal augmentation for  $T_H + IN + R$ .*

*Proof.* Since  $T_0 + A'_1 + A_2$  is a spanning subgraph of  $T_0 + A_1 + A_2$  and each edge of  $A_2$  is essential in  $T_0 + A_1 + A_2$  (by corollary 2), the edges in  $A_2$  are also essential in  $T_0 + A'_1 + A_2$ . With lemma 7 we conclude that  $A'_1 \cup A_2$  is a minimal augmentation for  $T_0$ . By lemma 3 (with  $H' = T_H + IN + R$  and  $H'' = T_0$ ) it follows that  $A'_1 \cup A_2$  is also a minimal augmentation for  $T_H + IN + R$ .  $\square$

### 3.3 Analysis

We now analyze the processor and time requirements of our algorithm on a PRAM. For the definitions of the various PRAM models we refer the reader to the survey paper by Karp and Ramachandran ([11]). One iteration of the while-loop of algorithm 2 can be implemented to run almost optimally in time  $O(\log n)$  on an ARBITRARY CRCW PRAM. Only the fact that no optimal algorithms are currently known for bucket sort and for computing connected components prevents us from achieving optimal performance. For sorting we use the algorithm of Bhatt et al. ([1]) which sorts  $n$  integers in the range  $0 \dots n^{O(1)}$  in time  $O(\log n / \log \log n)$  with  $n(\log \log n)^2 / \log n$  processors on an ARBITRARY CRCW PRAM. We need to compute connected components at several places in our algorithm (see below). The most efficient connectivity algorithm ([2]) computes the connected components of a graph with  $n$  nodes and  $m$  edges represented by adjacency lists in  $O(\log n)$  time with  $(m + n)\alpha(m, n) / \log n$  processors of an ARBITRARY PRAM; if the graph is represented by an unordered list of its edges, we can construct its adjacency list by sorting the edges lexicographically (using bucket sort in the range  $0 \dots n^{O(1)}$ ); this can be done in time  $O(\log n)$  with  $m \log \log n / \log n$  processors of an ARBITRARY PRAM.

Since more processor-efficient algorithms for these problems may be developed in the future, we shall adopt the following conventions:  $B(n)$  denotes the number of processors required to sort  $n$

$P_L$  is incident with at most 2 edges of  $A_2^L$ , we only need to look at at most two vertices in each 2-edge-connected component of  $T_0 + A_1$ . We also note that each edge in  $A_1^L$  has at least one endpoint that is a leaf of  $T_0$  and that is not incident with another edge of  $A_1^L$ . We call such an endpoint a *critical endpoint* of an edge of  $A_1^L$ . We observe that an edge in  $A_1^L$  can only be redundant in  $T_0 + A_1 + A_2$  if its critical endpoints are incident with edges in  $A_2^L$ . Thus, we only need to examine an edge of  $A_1^L$  if it is the unique edge in  $A_1^L$  incident with a leaf in  $T_0$  that is an endpoint of an edge of  $A_2^L$ . It follows that we only need to consider at most two edges of  $A_1^L$  in each 2-edge-connected component of  $T_0 + A_1$ . Thus, we may process the edges in  $A_1^L$  in two phases. In each phase we look at edges that lie in different 2-edge-connected components of  $T_0 + A_1$ . These edges certainly have edge-disjoint cycles in  $T_0$ . Hence, we may apply the idea described in the previous paragraph to remove a maximal set of edges in each phase.

The approach we have just outlined is essentially that taken by the following algorithm.

**Algorithm 5** Making  $A_1$  minimal.

*Input* Graph  $T_0$ , edge sets  $A_1$  and  $A_2$ .

*Output* Subset  $A_1' \subseteq A_1$  such that  $A_1' \cup A_2$  is a minimal augmentation for  $T_0$ .

- (0) Let  $H^* = T_0 + A_1 + A_2$ .
- (1) In parallel for each leaf chain  $L$ , do:
  - (1.1) In parallel for each vertex  $w$  on  $P_L$  incident with at least one edge of  $A_2^L$ , do:
    - (1.1.1) Let  $B_w$  be the 2-edge-connected component of  $T_0 + A_1$  that corresponds to  $w$  (possibly,  $B_w$  is a single vertex). Let  $V_w$  be the set of endpoints in  $B_w$  of the edges of  $A_2^L$ . (By observation 2, we have  $|V_w| \leq 2$ .)
    - (1.1.2) Process the vertices  $v$  of  $V_w$  sequentially as follows:
      - (1.1.2.1) If there is a unique edge  $e$  in  $A_1$  incident with  $v$ , remove  $e$  from  $H^*$ .
      - (1.1.2.2) If an edge on the fundamental cycle of  $e$  in  $T_0$  is a cutedge in  $H^*$ , put  $e$  back into  $H^*$ .
- (2) Let  $A_1'$  be the subset of edges in  $A_1$  that are contained in  $H^*$ .

**Lemma 7**  $T_0 + A_1' + A_2$  is 2-edge-connected and each edge of  $A_1'$  is essential in  $T_0 + A_1' + A_2$ .

*Proof.* The graph  $T_0 + A_1' + A_2$  is a spanning subgraph of  $T_0 + A_1 + A_2$ . To see that  $T_0 + A_1' + A_2$  is 2-edge-connected, note that an edge of  $A_1$  removed in step (1.1.2.1) is not put back in step (1.1.2.2) only if in  $H^*$  the edges of  $T_0$  that were covered by  $e$  are still covered (by some other edges). It

To see why  $A_2^L$  is minimal, let  $e_0 \dots e_k$  be the edges in  $A_2^L$  in the order they occur as vertices on the maximal path in  $D_L$ . By observation 1 and the definition of  $D_L$ , the edge from the lowest vertex (leaf) on  $P_L$  to its parent in  $T_1$  is covered only by  $e_0$ . Furthermore, the lower endpoint of  $e_{i+1}$  lies strictly above the upper endpoint of  $e_{i-1}$ . It follows that the edges on  $P_L$  between those endpoints are covered only by  $e_i$ . Finally, again by observation 1, the highest edge on  $P_L$  is covered only by  $e_k$ . Hence, the edges of  $A_2^L$  cover the edges of  $P_L$  minimally.  $\square$

**Corollary 2** *The graph  $T_0 + A_1 + A_2$  is 2-edge-connected and the edges in  $A_2$  are essential in this graph.*

*Proof.* By lemma 5 each non-stem edge of  $T_0$  is covered by some edge in  $A_1$ . By lemma 6 each stem edge of  $T_0$  is covered either by an edge of  $A_1$  or an edge of  $A_2$ . Hence,  $T_0 + A_1 + A_2$  is 2-edge-connected. By lemma 6 the set  $A_2$  is a minimal set such that  $T_1 + A_2$  is 2-edge-connected. By lemma 3  $A_2$  is also a minimal set such that  $T_0 + A_1 + A_2$  is 2-edge-connected. We conclude that the edges of  $A_2$  are essential in  $T_0 + A_1 + A_2$ .  $\square$

**Observation 2** *Every vertex of  $P_L$  is incident with at most 2 edges of  $A_2^L$ .*

The set  $A_1 \cup A_2$  may not be a minimal augmentation for  $T_0$ . Indeed, an edge of  $A_1$  is redundant in  $T_0 + A_1 + A_2$  if all the tree edges on its fundamental cycle in  $T_0$  are covered by other edges in  $A_1 \cup A_2$ .

To obtain a minimal augmentation for  $T_0$ , we shall remove a subset of the edges in  $A_1$  from  $T_0 + A_1 + A_2$  such that the resulting subgraph of  $T_0 + A_1 + A_2$  is 2-edge-connected and all edges of  $A_1 \cup A_2$  contained in this subgraph are essential in this subgraph.

We note that an edge  $e$  in  $A_1$  can be removed from  $T_0 + A_1 + A_2$  (without destroying 2-edge-connectivity) if no edge on the fundamental cycle of  $e$  in  $T_0$  is a cutedge in the resulting graph. We may extend this idea to a subset  $B \subseteq A_1$  with the property that the fundamental cycles of the edges in  $B$  are edge-disjoint: we may remove exactly those edges  $e$  in  $B$  from  $T_0 + A_1 + A_2$  with the property that none of the edges on the fundamental cycle of  $e$  in  $T_0$  is a cutedge in  $T_0 + A_1 + A_2 - B$ .

Note that edges in  $A_1$  whose lca's lie on different leaf chains of  $T$  (the tree representing  $T_{lca}$  at the current stage of tree contraction) have edge-disjoint fundamental cycles in  $T_0$ . Thus, we may apply the idea from the previous paragraph to process edges of  $C$  whose lca's belong to different leaf chains independently of each other. Let  $A_1^L$  contain those edges in  $A_1$  whose lca lies on leaf chain  $L$ . It suffices to show how to remove a maximal subset of edges in  $A_1^L$  so that the remaining edges are essential. Note that edges in  $A_1^L$  may have overlapping fundamental cycles in  $T_0$ . Thus, it is not sufficient (in general) to remove all the edges in  $A_1^L$  from  $T_0 + A_1 + A_2$  at one time and check the resulting graph for cutedges. Here is where observation 2 comes in: since each vertex on



In parallel for each leaf chain  $L$ , do:

- (1) Let  $C_L$  be the subset of edges in  $C - A_1$  that connect two distinct vertices on  $P_L$ . For each vertex  $v$  on  $P_L$  let  $E_v$  be the set of edges in  $C_L$  incident with  $v$  and whose other endpoint lies strictly above  $v$  on  $P_L$  (i.e., closer to the root of  $T_1$ ). Each vertex  $v$  on  $P_L$  with  $E_v \neq \emptyset$  selects a single edge in  $E_v$  whose upper endpoint is closest to the root of  $T_1$ . Denote this edge by  $up(v)$  (see figure 4(a)).
- (2) Construct the auxiliary digraph  $D_L$  defined as follows (see figure 4(b)): the vertex set of  $D_L$  is the set  $\{up(v) : v \text{ is a vertex on } P_L \text{ and } E_v \neq \emptyset\}$ . There is a directed edge in  $D_L$  from  $e$  to  $e'$  if and only if the lower endpoint of  $e'$  lies between the endpoints of  $e$ , possibly coinciding with the upper endpoint of  $e$ , and the upper endpoint of  $e'$  is strictly above that of  $e$  (i.e, it is closer to the root of  $T_1$ ); moreover, among all edges whose lower endpoint lies between the endpoints of  $e$ , the upper endpoint of  $e'$  is closest to the root of  $T_1$ . (Note: There may be several such edges  $e'$ ; in this case pick an arbitrary such edge  $e'$ . Hence, each vertex in  $D_L$  has outdegree at most 1.)
- (3) Starting at  $up(v)$ , where  $v$  is the unique leaf on  $P_L$ , construct a maximal path in  $D_L$ , i.e., a path that ends at a vertex that has outdegree 0 in  $D_L$ . Let  $A_L$  be the set of edges in  $C_L$  that are represented by vertices on this path.

Let  $A_2 = \bigcup_L A_2^L$

**Observation 1** *The lower endpoints on path  $P_L$  of edges in  $A_2^L$  are all distinct. Furthermore, the upper endpoints on  $P_L$  of the edges in  $A_2^L$  are all distinct.*

**Lemma 6**  *$A_2$  is a minimal set of edges covering the edges of  $T_1$ .*

*Proof.* As mentioned earlier, edges in  $C - A_1$  have their endpoints in  $T_1$  on the same path  $P_L$  from the root to some leaf of  $T_1$ . Therefore, it suffices to show that the edges in  $A_2^L$  minimally cover the edges of  $P_L$  for an arbitrary leaf chain  $L$  of  $T$ . We shall first prove that each edge in  $P_L$  is covered by some edge in  $A_2^L$ . Assume for a contradiction that  $e$  is the lowest edge on  $P_L$  not covered by any edge in  $A_2^L$ . Consider the subset  $S$  of edges of  $C - A_1$  that cover  $e$ . Since  $H$  is 2-edge-connected,  $S$  is nonempty. Let  $e'$  be an edge of  $S$ . Since  $e$  is the lowest uncovered edge on  $P_L$ , the lower endpoint of  $e'$  lies between the endpoints of some edge  $e'' \in A_2^L$  (may coincide with its upper endpoint). The path in  $D_L$  found in step (3) is maximal. Therefore,  $e''$  has a successor edge on the path whose upper endpoint is at least as high on the stem as that of  $e'$ . Hence, that edge covers  $e$ , contradicting the assumption that  $e$  is not covered.

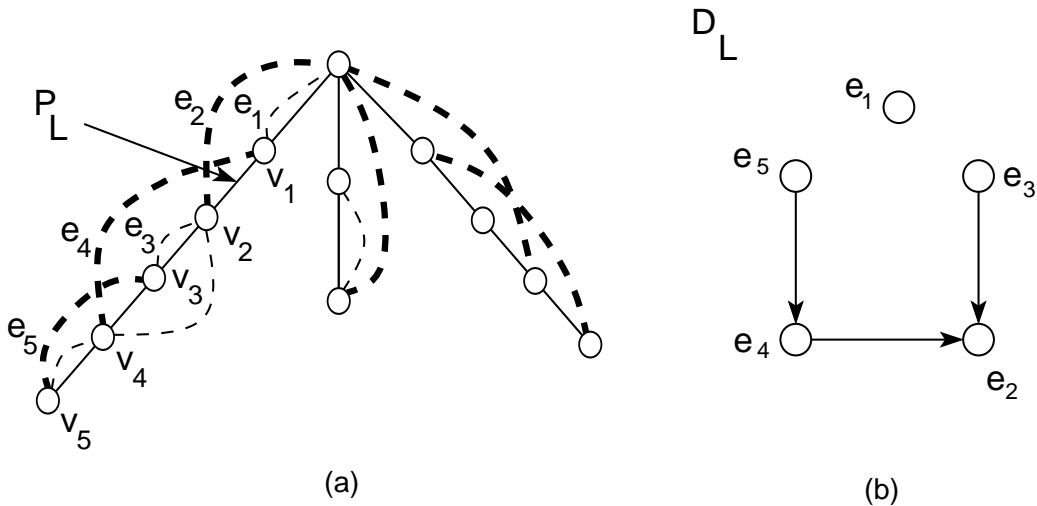


Figure 4: (a) Condensation  $T_1$  of  $T_H + IN + R + A_1$ . Solid lines correspond to edges in  $T_1$ . Dashed lines correspond to nontree edges in  $H$ . Thick dashed lines represent edges in  $A_2$ . On path  $P_L$  we have  $up(v_i) = e_i$ . (b) Graph  $D_L$  corresponding to root-to-leaf path  $P_L$  in  $T_1$ . Edges  $e_5, e_4$  and  $e_2$  belong to  $A_2$  because they represent vertices on a maximal path in  $D_L$  starting at  $e_5$ .

root of  $T_1$  has at most one child. Moreover, the edges on any root-to-leaf path in  $T_1$  belong to the stem of a single leaf chain of  $T$ . The tree  $T_1$  corresponding to the example in figure 3 is shown in figure 4(a).

Let  $P_L$  denote the root-to-leaf path in  $T_1$  containing edges on the stem of leaf chain  $L$  of  $T$ . We note that all the nontree edges in  $C$  connect vertices on the same path  $P_L$  for some leaf chain  $L$  (this follows from the definition of  $C$ ). Let  $C_L$  be the set of edges in  $C$  that connect two distinct vertices on  $P_L$ . It suffices to show how to compute a minimal subset of edges in  $C_L$  that covers the edges on path  $P_L$ . For any edge of  $C_L$ , let the *upper endpoint* refer to the endpoint that is closer to the root of  $T_1$  and let the *lower endpoint* be that endpoint that is further away from the root of  $T_1$ . The following greedy procedure picks a minimal subset of edges covering the edges on  $P_L$ : pick as first edge an edge of  $C_L$  incident with the unique leaf on  $P_L$  and whose endpoint is closest to the root of  $T_1$ . At any point pick as the next edge an edge in  $C_L$  whose lower endpoint lies between the endpoints of the edge that was picked last and whose upper point is closest to the root of  $T_1$ . We are done once we pick an edge whose upper endpoint is the root of  $T_1$ . The following algorithm is a parallel version of this greedy strategy.

**Algorithm 4** Covering  $T_1$ .

*Input* Tree  $T_1$ , edge set  $C - A_1$ .

*Output* Minimal subset  $A_2 \subseteq C - A_1$  covering the edges in  $T_1$ .

that  $e$  is a non-stem edge in  $T_0$ . Suppose that  $e$  links a node  $v'$  in  $T_H$  to its parent in  $T_H$ . By the definition of  $T_0$  and  $R$ , no edge of  $R$  is incident with a descendant of  $v'$  in  $T_H$  (otherwise the endpoints of  $e$  would have been collapsed into a single vertex of  $T_0$ ). Furthermore, if an lca of an edge in  $IN$  is a descendant of  $v'$ , it must be a proper descendant. Hence  $v = v'$  and  $v$  is a leaf of  $T_H$ . It follows that  $v$  is not an lca of an edge in  $C$ . Hence,  $e$  is a non-stem edge.

Now let each leaf of  $T_0$  be incident with an edge of  $A_1$ . Let  $e$  be a non-stem edge of  $T_0$  connecting some vertex  $v$  to its parent in  $T_0$ . Let  $D$  be the set of edges in  $A_1$  incident with leaves in  $T_0$  that are descendants of  $v$ . By our assumption  $D$  is nonempty. Moreover, any edge in  $D$  has its lca on a stem in  $T_H$ ; hence its fundamental cycle must contain edge  $e$ . We conclude that each non-stem edge in  $T_0$  is covered by an edge of  $A_1$ .  $\square$

**Lemma 5**  $A_1$  is a minimal subset of edges in  $C$  covering all non-stem edges of  $T_0$ .

*Proof.* By lemma 4 we only need to show that  $A_1$  is a minimal set of edges such that each leaf of  $T_0$  is incident with an edge in  $A_1$ . We first show that each leaf of  $T_0$  is incident with some edge in  $A_1$ . If  $v$  is a leaf in tree  $T_l$  of the forest, then its unique incident tree edge marked in step (2.1) will never be unmarked. Now assume that  $v$  is not a leaf. If its depth is even, it will certainly be incident with a marked edge after step (2.3). If its depth is odd, the edge to its parent in  $T_l$  can be unmarked only if one of its children is a leaf. The edge to that child is marked in step (2.1) and will never be unmarked. Finally, any isolated vertices in  $G_e$  (all of whose incident edges of  $C$  connect to nodes that are not leaves of  $T_0$ ) will be incident with a marked edge after step (3). Hence, every leaf in  $T_0$  is incident with an edge of  $A_1$ .

To establish the minimality of  $A_1$ , fix an edge in  $A_1$ . If it was marked in step (3), then only one endpoint of this edge is a leaf of  $T_0$  and this endpoint will lie on a single marked edge. For the remaining edges of  $A_1$ , observe that edges that were bad after step (2.1) were removed during step (2.2) and none of the edges marked in step (2.3) are bad. Hence, every edge of  $A_1$  has at least one endpoint that is a leaf in  $T_0$  and that is not incident with another edge of  $A_1$ . Therefore  $A_1$  is minimal.  $\square$

In stage 1 we covered all non-stem edges of  $T_0$  and possibly some stem edges. In the second stage we proceed to cover stem edges in  $T_0$  that have not been covered in stage 1. The algorithm we use here is similar to an algorithm in [17, 16] for finding a minimum feedback-vertex-set in a reducible flow graph.

To compute a minimal subset  $A_2 \subseteq C$  covering the stem edges of  $T_0$  not covered by edges of  $A_1$ , we appeal again to lemma 3. According to this lemma we may choose for  $A_2$  a minimal set of edges in  $C$  covering the edges in the condensation  $T_1$  of  $T_0 + A_1$ . Note that  $T_1$  is also the condensation of graph  $T_H + IN + R + A_1$ . The tree  $T_1$  has a rather simple structure: any vertex other than the

Furthermore, by the minimality of  $A_2$ , all edges of  $A_2$  are essential in  $T_0 + A_1 + A_2$ . To obtain a minimal augmentation for  $T_0$ , we shall remove certain edges of  $A_1$  from  $T_0 + A_1 + A_2$  so that the resulting subgraph of  $T_0 + A_1 + A_2$  is 2-edge-connected and all remaining nontree edges in  $A_1 \cup A_2$  are essential in this subgraph. This is done in the third and final stage.

One can show (lemma 4) that a set of nontree edges  $A_1 \subseteq C$  covers all non-stem edges of  $T_0$  if and only if each leaf of  $T_0$  is incident with at least one edge of  $A_1$ . Thus, in stage 1 we need to compute a minimal set  $A_1 \subseteq C$  such that each leaf of  $T_0$  is incident with at least one edge of  $A_1$ . This is exactly what algorithm 3 does. In figure 3 the thick dashed lines represent nontree edges that are put into  $A_1$ . Note that the set of these edges is minimal with respect to the property that each leaf of  $T_0$  is incident with an edge in this set.

**Algorithm 3** Covering non-stem edges of  $T_0$ .

*Input* Tree  $T_0$ , set  $C$  of nontree edges.

*Output* Minimal subset  $A_1 \subseteq C$  covering the non-stem edges of  $T_0$ .

- (1) Let  $G_l$  be the graph whose vertices are the leaves of  $T_0$  and whose edges are the edges of  $C$  between them. Find a spanning forest in  $G_l$ .
- (2) For each tree  $T_l$  in this forest do the following:
  - (2.1) Root the tree at an arbitrary vertex. Determine the depth of each node in  $T_l$ . Mark all edges in  $T_l$  that connect a vertex at even depth with a child (at odd depth). Also, for each leaf in  $T_l$  mark the unique incident tree edge (if it is not already marked).
  - (2.2) A marked edge in the tree is *bad* if both of its endpoints have at least two marked edges (including this edge) incident with them. Each node (at even depth) eliminates any bad edges to its children.
  - (2.3) If a node at even depth loses all marked edges to its children in the previous step, it marks a single edge to one of its children in  $T_l$ .
- (3) Any leaf of  $T_0$  that is not yet incident with a marked edge (i.e., it is an isolated vertex of  $G_l$ ) marks an arbitrary edge of  $C$  incident with it.

Let  $A_1$  be the subset of edges in  $C$  that are marked.

**Lemma 4**  $A_1$  covers all non-stem edges of  $T_0$  if and only if each leaf of  $T_0$  is incident with an edge of  $A_1$ .

*Proof.* Suppose that  $A_1$  covers the non-stem edges of  $T_0$ . Let  $v$  be a leaf of  $T_0$  and let  $e$  be the edge from  $v$  to its parent in  $T_0$ . To show that an edge of  $A_1$  is incident with  $v$ , it suffices to show

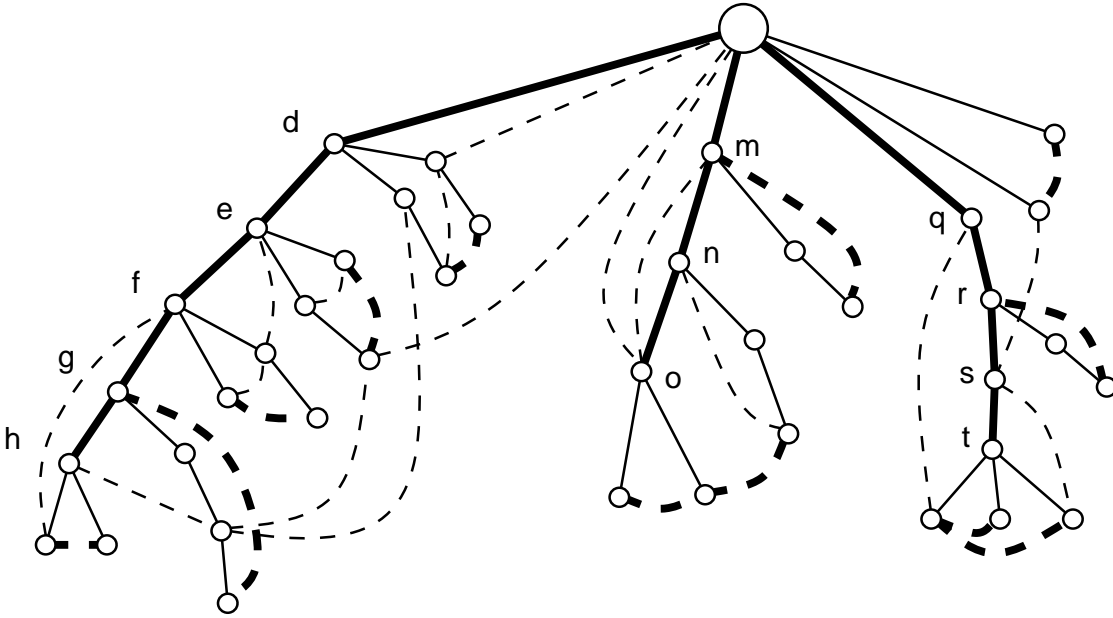


Figure 3: Condensation  $T_0$  of  $T_H + IN + R$ . The edges of  $T_0$  are indicated by solid lines. Thick solid lines indicate stem edges. Dashed lines correspond to nontree edges in  $H$ . Thick dashed lines represent nontree edges that are put into  $A_1$ .

Let us return to our original problem. We want to compute a minimal augmentation  $A \subseteq C$  for  $T_H + IN + R$ . Let  $T_0$  denote the condensation of  $T_H + IN + R$ . By lemma 3 we may choose for  $A$  a minimal augmentation for  $T_0$ . We shall illustrate our method to compute such a minimal augmentation with the example in figure 2. Consider the first iteration of algorithm 2 on the graph  $H$  depicted in figure 2(a). In this case  $IN = \emptyset$  and  $R$  is the set of nontree edges whose lca is in the set  $\{a, b, j, k\}$ . The condensation  $T_0$  of  $T_H + IN + R$  is indicated in figure 3 by solid lines. The dashed lines represent nontree edges in  $C$ .

Our method for computing a minimal augmentation for  $T_0$  is based on partitioning the edges in  $T_0$  into two classes. Recall that tree  $T$  represents  $T_{lca}$  at the current stage of tree contraction (it will change in step (2.4) at the end of this iteration of the while-loop in algorithm 2). Let  $L = \langle v_0, \dots, v_k \rangle$  be a leaf chain of  $T$ . The path in  $T_H$  from  $v_0$  to  $v_k$  is termed the *stem of  $L$* . We call those edges in  $T_0$  that belong to a stem of a leaf chain *stem edges* and refer to the remaining edges as *non-stem edges*. In figure 3 the stem edges are indicated by thick solid lines and nonstem edges are indicated by thin solid lines.

We build a minimal augmentation for  $T_0$  in three stages: in stage 1 we select a minimal subset  $A_1 \subseteq C$  covering the non-stem edges in  $T_0$ . In stage 2 we choose a minimal subset  $A_2 \subseteq C$  whose edges cover all stem edges in  $T_0$  not covered in stage 1. The graph  $T_0 + A_1 + A_2$  is 2-edge-connected.

In the sequel we consider a fixed iteration of the while-loop in algorithm 2. We have reduced the problem of computing a minimal augmentation for  $T_H$  to the problem of computing a minimal augmentation for  $T_H + IN + R$  from the edges in  $C$ , i.e., a minimal set  $A \subseteq C$  such that  $T_H + IN + R + A$  is 2-edge-connected. At this point it is not clear that this problem is easier than the original problem of computing a minimal augmentation for  $T_H$ . We shall see, however, that the special structure of  $C$ , i.e., the fact that all edges in  $C$  have their lca on a leaf chain of  $T$ , allows for a fast parallel solution of the modified augmentation problem. Intuitively, this is because nontree edges whose lca's lie on different leaf chains can be processed independently.

It is helpful to view the minimal augmentation problem in terms of edges in the spanning tree  $T_H$  being *covered* by nontree edges. To make this precise, note that any nontree edge  $e$  in  $H$  produces a unique cycle in  $T_H + e$ , the *fundamental cycle* of  $e$ . We say that edge  $e$  *covers* the edges of  $T_H$  that lie on its fundamental cycle. We say that a set  $D$  of nontree edges covers a set  $D'$  of tree edges if each tree edge in  $D'$  is covered by some edge in  $D$ . The definition of 2-edge-connectivity implies that  $A$  is a minimal augmentation for  $T_H + IN + R$  if  $A$  is a minimal set of nontree edges in  $C$  covering the edges in  $T_H$  not already covered by an edge in  $IN \cup R$ .

Since we are only interested in the edges of  $T_H$  not covered by edges in  $IN \cup R$ , we would like to somehow “remove” those edges in  $T_H$  that are covered by an edge in  $IN \cup R$ . This motivates the following definition. Let  $H'$  be a subgraph of  $H$  containing tree  $T_H$  as a subgraph. The *condensation of graph  $H'$*  is a graph  $H''$  obtained from  $H'$  by collapsing the vertex sets of the 2-edge-connected components of  $H'$ . (The operation of “collapsing” is defined formally in section 2.) Note that  $H''$  is a tree. We root this tree at the vertex into which the root of  $T_H$  is collapsed.

The following lemma shows that we can reduce the problem of minimally augmenting  $H'$  to the problem of minimally augmenting its condensation  $H''$ .

**Lemma 3** *Let  $H'$  be a subgraph of  $H$  containing  $T_H$  and let  $H''$  be the condensation of  $H'$ . For any  $B \subseteq E(H)$ ,  $H' + B$  is 2-edge-connected iff  $H'' + B$  is 2-edge-connected.*

*Proof.* The edges of  $H''$  are the cutedges of  $H'$ . Moreover, a cycle in  $H' + B$  containing a cutedge of  $H'$  maps to a cycle in  $H'' + B$  containing the corresponding edge of  $H''$ . If  $H' + B$  is 2-edge-connected, then every cutedge of  $H'$  is on a cycle in  $H' + B$  and hence, every edge of  $H'' + B$  is on a cycle. Therefore,  $H'' + B$  is 2-edge-connected.

Conversely, we note that a cycle in  $H'' + B$  yields a cycle in  $H' + B$  containing all the cutedges of  $H'$  that lie on the cycle of  $H'' + B$ . Thus, if  $H'' + B$  is 2-edge-connected, every edge of  $H''$  is on a cycle in  $H'' + B$  and hence, every cutedge of  $H'$  is on a cycle in  $H' + B$ . Therefore,  $H' + B$  is 2-edge-connected.  $\square$

stage, and those with lca  $a$  are processed at the third and last stage.

For technical reasons we view the tree consisting of a single vertex as having a leaf chain with no root vertex; one more iteration of tree contraction will yield the empty tree. The fine-structure of step (2.2) will be developed following lemma 2. Note that a stage as explained earlier corresponds to an iteration of the while-loop.

**Algorithm 2** Finding a minimal augmentation for  $T_H$ .

*Input* Tree  $T_H$ , set  $F$  of nontree edges.

*Output* Minimal augmentation  $IN$  for  $T_H$  from edges in  $F$ .

- (1) Initialize :  $IN := \emptyset$ ,  $R := F$ , and  $T := T_{lca}$ .
- (2) While  $T$  is non-empty, perform the following actions for all leaf chains of  $T$  in parallel:
  - (2.1) Let  $C$  be the subset of edges in  $R$  whose lca is a vertex on some leaf chain of  $T$ . Let  $R := R - C$ .
  - (2.2) Find a minimal subset  $A \subseteq C$  such that  $T_H + IN + R + A$  is 2-edge-connected.
  - (2.3) Let  $IN := IN \cup A$ .
  - (2.4) Remove the leaf chains from  $T$ .

**Lemma 2** Upon completion of algorithm 2,  $IN$  is a minimal augmentation for  $T_H$ .

*Proof.* We claim that the following statement holds before any iteration of step (2):

- (\*)  $T_H + IN + R$  is 2-edge-connected and every edge of  $IN$  is essential in  $T_H + IN + R$ .

Upon termination of algorithm 2,  $R$  is empty and (\*) implies that  $IN$  is a minimal augmentation for  $T_H$ .

We prove that (\*) holds before each iteration of step (2) by induction on the iteration number. Note that (\*) holds before the first iteration where  $R = F$  and  $IN = \emptyset$  since  $H = T_H + F$  is 2-edge-connected. Assume that (\*) holds before the  $i$ th iteration of step (2) ( $i \geq 1$ ). Denote by  $IN_j$  and  $R_j$  the sets  $IN$  and  $R$ , respectively, before the  $j$ th iteration for any  $j \geq 1$ . By step (2.2) in the  $i$ th iteration  $T_H + IN_i + R_{i+1} + A$  is 2-edge-connected, and hence, by step (2.3),  $T_H + IN_{i+1} + R_{i+1}$  is 2-edge-connected. Note that  $IN_{i+1} \cup R_{i+1} \subseteq IN_i \cup R_i$ . Thus, by the inductive assumption, the edges of  $IN_i$  are essential in  $T_H + IN_{i+1} + R_{i+1}$ . By the minimality of the set  $A$  constructed in step (2.2) each edge of  $IN_{i+1} - IN_i$  is essential in  $T_H + IN_{i+1} + R_{i+1}$ . This shows that (\*) holds before iteration  $i + 1$  of step (2).  $\square$

indicated by dashed lines. We shall make use of *parallel tree contraction* to compute such a minimal augmentation. Tree contraction was first introduced by Miller and Reif ([15]). We use a variant of tree contraction proposed in [17, 16]. This method is based on the operation *Shrink* which we shall now describe.

Let  $T$  be an arbitrary rooted tree. A *leaf chain* in  $T$  is a sequence  $\langle v_0, v_1, \dots, v_k \rangle$  of vertices in  $T$  such that  $v_k$  is a leaf,  $v_i$  is the unique child of  $v_{i-1}$  for  $i > 0$  and the parent  $v$  of  $v_0$  is either the root of  $T$  or has at least 2 children. For example consider the tree in figure 2(b). This tree has one leaf chain with 6 nodes ( $\langle c, d, e, f, g, h \rangle$ ), one leaf chain with 5 nodes ( $\langle p, q, r, s, t \rangle$ ), one leaf chain with 4 nodes ( $\langle l, m, n, o \rangle$ ) and one leaf chain with 1 node ( $\langle i \rangle$ ).

The operation *Shrink* reduces a tree by removing the vertices in the leaf chains. Tree contraction reduces an arbitrary rooted tree to a single vertex by repeatedly applying the *Shrink* operation to it. It can be shown ([17, 16]) that  $O(\log n)$  applications of *Shrink* contract any tree on  $n$  nodes to a single vertex. Applying *Shrink* to the tree in figure 2(b) removes vertices  $c \dots h, i, l \dots o, p \dots t$  and applying it one more time removes vertices  $b, j, k$ , thus leaving a tree with the single vertex  $a$ .

In the sequel we assume that the spanning tree  $T_H$ , computed in step (1.1) of algorithm 1, is rooted at an arbitrary vertex. The tree contraction will be performed on a tree  $T_{lca}$  closely related to the tree  $T_H$ . The tree is defined as follows: the vertices of  $T_{lca}$  are the least common ancestors (*lca*'s) in  $T_H$  of nontree edges in  $H$  (i.e., the *lca*'s of the endpoints of those edges). The root of  $T_{lca}$  is the root of  $T_H$ . A vertex  $v$  is a child of  $u$  in  $T_{lca}$  if and only if  $v$  is a descendant of  $u$  in  $T_H$  and none of the internal vertices on the path from  $u$  to  $v$  in  $T_H$  is an *lca* of a nontree edge. For instance, consider the graph  $H$  depicted in figure 2(a) with the spanning tree  $T_H$  denoted by the solid edges. The corresponding tree  $T_{lca}$  is the tree in figure 2(b).

Let  $F$  denote the set of nontree edges in  $H$ , i.e., those edges in  $H$  that do not belong to the spanning tree  $T_H$ . Let  $T$  denote the tree  $T_{lca}$  at the current stage of tree contraction. The algorithm for computing a minimal augmentation for  $T_H$  works in stages. In each stage the algorithm maintains the following three sets of nontree edges (subsets of  $F$ ): the set  $IN$  of edges that have been processed at an earlier stage and have been chosen for the minimal augmentation; the set  $C$  of those nontree edges whose *lca* lies on a leaf chain of  $T$ ; and the set  $R$  of nontree edges that are still to be processed. At the current stage we add to the augmentation a subset  $A$  of  $C$  that is minimal with respect to the property that  $T_H + IN + R + A$  is 2-edge-connected. Thus, we add a subset of edges from  $C$  to the augmentation while taking into account those edges that have already been added as well as those that will be examined at later stages. This will ensure that the resulting set of edges for the augmentation is indeed minimal. This will be proven formally in lemma 2.

For the example in figure 2(a) nontree edges with an *lca* other than  $a, b, j, k$  will be processed at the first stage of our algorithm (algorithm 2), those whose *lca* is  $b, k$  or  $j$  are processed at the second



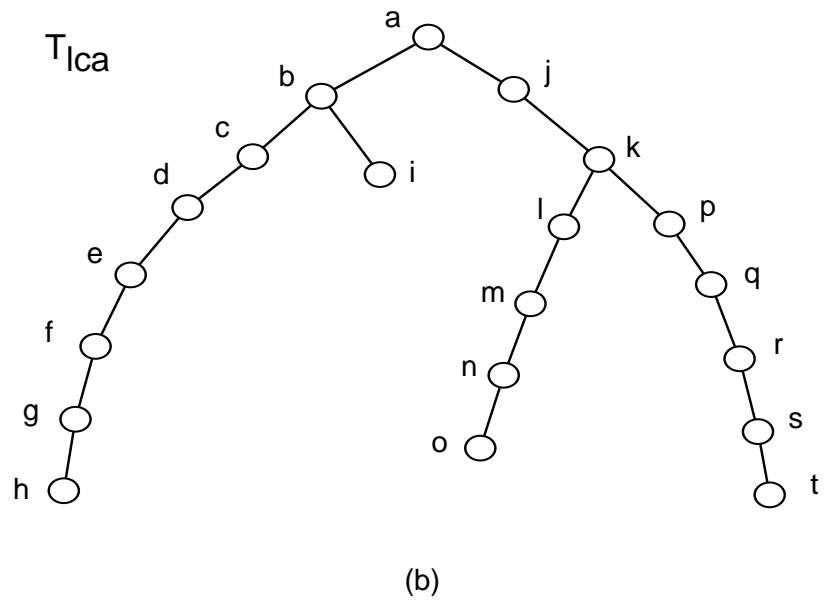
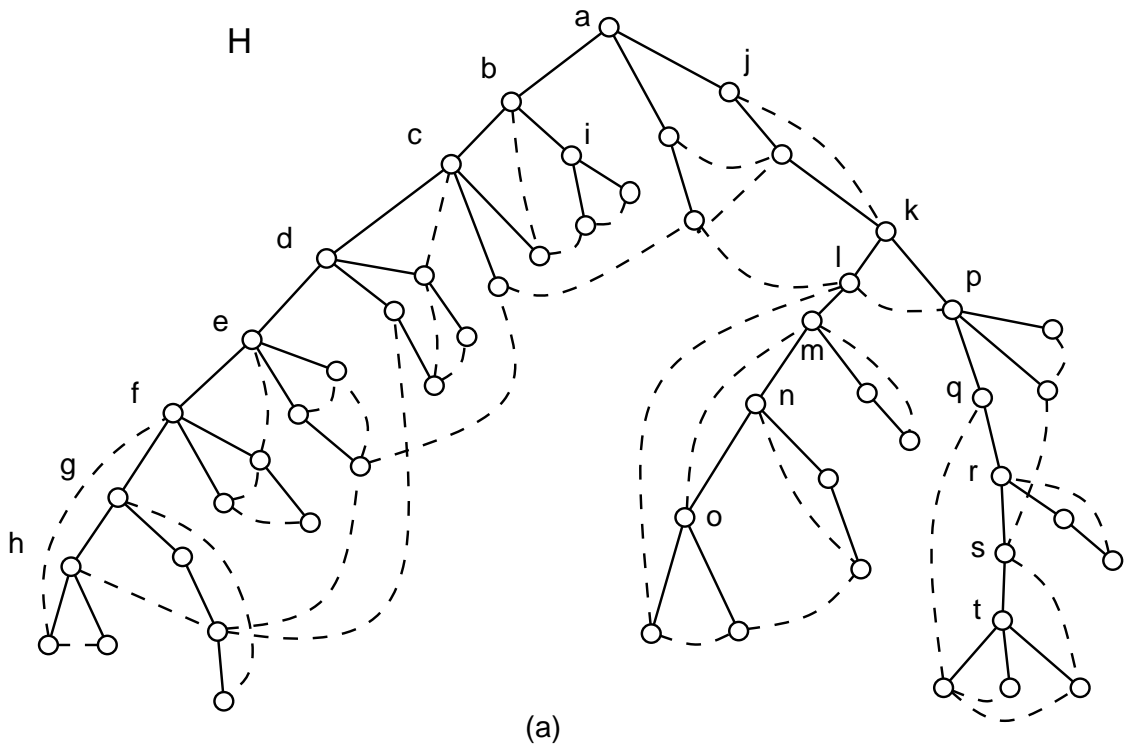


Figure 2: (a) Graph  $H$  with the edges of  $T_H$  (rooted at  $a$ ) indicated by solid lines and the non-tree edges indicated by dashed lines. Vertices with a letter are lca's of nontree edges. (b) The corresponding tree  $T_{lca}$ .

mentation is similar to our method but it does not require redundant and essential edges to be computed explicitly.

The correctness of algorithm 1 follows from these two observations: (1) at the start of any iteration of the while-loop (step (1) of algorithm 1) the graph  $H$  is a spanning 2-edge-connected subgraph of  $G$ ; (2) the number of edges in  $H$  strictly decreases in each iteration of the while-loop.

The following lemma implies that the number of redundant edges in  $H$  decreases quite rapidly. This results in a  $O(\log n)$  upper bound on the number of iterations of algorithm 1.

**Lemma 1** *There is a spanning tree in  $H$  that contains at most  $2/3$  of the redundant edges in  $H$ .*

*Proof.* Let  $Ess$  denote the set of essential edges in  $H$ . Let  $r$  denote the number of redundant edges in  $H$  and  $c$  the number of connected components of the graph  $(V(H), Ess)$ . Fix a spanning tree  $T$  in  $H$  with the smallest possible number of redundant edges of  $H$ . The tree  $T$  contains exactly  $c - 1$  redundant edges of  $H$ . Furthermore, if  $c > 1$ , then each connected component of  $(V(H), Ess)$  is incident with at least 3 redundant edges in  $H$  and hence,  $c \leq 2/3 \cdot r$ . The claim follows.  $\square$

**Corollary 1** *Algorithm 1 terminates after  $O(\log n)$  iterations of the while-loop.*

*Proof.* Let  $H'$  and  $H''$  represent the graph  $H$  at the start of two consecutive iterations of the while-loop. Note that all edges in  $H''$  added to the spanning tree  $T_{H'}$  in step (1.2) of the current iteration are essential in  $H''$ . Thus, any redundant edge in  $H''$  is redundant in  $H'$  and belongs to  $T_{H'}$ . The previous lemma then implies that the number of redundant edges in  $H''$  is at most  $2/3$  times the number of redundant edges in  $H'$ . The claim follows.  $\square$

The main work in algorithm 1 is done in step (1.2) in which we minimally augment the spanning tree  $T_H$  into a 2-edge-connected graph. We call the (minimal) set  $B$  of edges in  $H$  that will be added to  $T_H$  a *minimal augmentation* for  $T_H$  in  $H$ . In the next section we describe how such a minimal augmentation can be computed efficiently in parallel.

### 3.2 Computing a Minimal Augmentation in Parallel

For this section we fix an iteration of the while-loop of algorithm 1. We describe how to compute a minimal augmentation for the tree  $T_H$  computed in step (1.1) of the current iteration.

Our method does not use the fact that  $T_H$  contains a minimum number of redundant edges in  $H$ . We may therefore assume that  $T_H$  is an arbitrary spanning tree in  $H$ . We shall illustrate our minimal augmentation procedure with the example in figure 2(a). Figure 2(a) shows a graph  $H$  with a spanning tree  $T_H$ ; the edges of  $T_H$  are indicated by solid lines and the nontree edges are

essential in any 2-edge-connected spanning subgraph of  $H$ . Similarly an edge that is redundant in a 2-edge-connected spanning subgraph of  $H$  is also redundant in  $H$ . In figure 1 essential and redundant edges are indicated by thick and thin lines, respectively. Since the graph in figure 1(b) is a minimal 2-edge-connected spanning subgraph of the graph in figure 1(a), all of its edges are essential.

Let  $G = (V, E)$  be a 2-edge-connected graph with  $n$  vertices and  $m$  edges. We present our algorithm for finding a minimal 2-edge-connected spanning subgraph of  $G$  in a top-down fashion. At the highest level it has the following structure:

**Algorithm 1** Finding a minimal 2-edge-connected spanning subgraph of  $G$ .

*Input* 2-edge-connected graph  $G$ .

*Output* Minimal 2-edge-connected spanning subgraph  $H$  of  $G$ .

- (0) Find a 2-edge-connected spanning subgraph  $H$  of  $G$  with fewer than  $2n$  edges.
- (1) While  $H$  contains redundant edges, repeat the following two steps:
  - (1.1) Find a spanning tree in  $H$  that contains the smallest possible number of redundant edges in  $H$ . Call this tree  $T_H$ .
  - (1.2) Determine a minimal subset  $B$  of edges in  $H$  such that the graph  $T_H + B$  is 2-edge-connected. Let  $H = T_H + B$ .

The purpose of step (0) is to speed up subsequent iterations of the while-loop by computing a sparse subgraph of the input graph. In this step we compute an ear decomposition of  $G$  ([14], [13], [18]) and eliminate all trivial ears. Let  $H$  be the resulting graph.  $H$  is clearly a 2-edge-connected spanning subgraph of  $G$ . Let  $m'$  denote the number of edges of  $H$  and let  $q$  be the number of (nontrivial) ears in the above ear decomposition. A proof by induction over  $q$  establishes  $m' = n + q - 2$ . Since  $H$  contains no trivial ears, we have  $q \leq n - 1$ ; hence,  $m' \leq 2n - 3$  as required in step (0).

For steps (1) and (1.1) we need to determine the redundant edges in  $H$ . A *separating pair of edges for  $H$*  is a pair  $(e_1, e_2)$  of edges of  $H$  such that the graph  $H - e_1 - e_2$  is not connected. Note that an edge of  $H$  is redundant if and only if it does not occur in any separating pair of edges for  $H$ . Hence, we identify the redundant edges in  $H$  by finding all separating pairs of edges in  $H$ . For this we modify the vertex triconnectivity algorithm in [3], [18] (see also [19]). We now assign weight 0 to essential edges and weight 1 to redundant edges and choose for  $T_H$  a minimum spanning tree in this graph. The implementation of step (1.2) will be discussed in the next section.

Note: An alternative (sequential) method developed independently by Han and Tarjan ([8]) for computing a minimal 2-edge-connected spanning subgraph can be parallelized; the parallel imple-

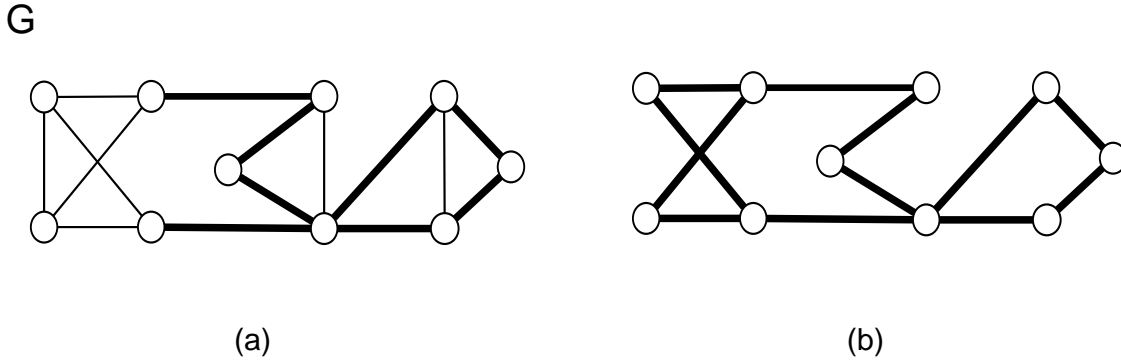


Figure 1: (a) A 2-edge-connected graph  $G$  and (b) a minimal 2-edge-connected spanning subgraph of  $G$ . Essential and redundant edges are indicated by thick and thin lines, respectively.

maximal biconnected subgraph of  $G$ .

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

### 3 Finding a Minimal 2-Edge-Connected Spanning Subgraph

In this section we consider the following problem: given a 2-edge-connected graph  $G$ , find a minimal 2-edge-connected spanning subgraph of  $G$ , i.e., a 2-edge-connected spanning subgraph of  $G$  that does not have a 2-edge-connected spanning subgraph of  $G$  as a proper subgraph. Figure 1 shows a 2-edge-connected graph  $G$  and a minimal 2-edge-connected spanning subgraph of  $G$ .

#### 3.1 The High-Level Algorithm

Our parallel algorithm for computing a minimal 2-edge-connected spanning subgraph makes use of a partition of the edges of a 2-edge-connected graph into two classes: those that can be removed without destroying 2-edge-connectivity and those whose removal destroys 2-edge-connectivity. Formally, for an arbitrary 2-edge-connected graph  $H$ , we say that an edge  $e$  is *redundant in  $H$*  if  $H - e$  is 2-edge-connected; edges that are not redundant in  $H$  are *essential in  $H$* . We sometimes shall not mention the graph  $H$  if it is clear from the context. Note that an edge that is essential in  $H$  is also

A *rooted tree* is a directed graph whose undirected version is a tree, having one vertex, called the *root*, which is the head of no edges, and such that all vertices except the root are the head of exactly one edge. If  $(v, w)$  is an edge in the rooted tree,  $v$  is the *parent* of  $w$  and  $w$  is the *child* of  $v$ . A *leaf* in the rooted tree is a vertex in the tree that does not have a child. A *descendant* of  $v$  is any node reachable from  $v$  in the tree (including  $v$ ). A vertex  $w$  is an *ancestor* of  $v$  if  $v$  is a descendant of  $w$  in the tree. A *proper descendant* (ancestor) of  $v$  is a descendant (ancestor) of  $v$  other than  $v$ . The *depth* of a node in a rooted tree is the number of edges on the (unique) simple path from the root to the node in the tree. The *least common ancestor* (lca) of two vertices in a rooted tree is the vertex at maximum depth that is an ancestor of both vertices.

A *spanning tree* of a graph  $G$  is a spanning subgraph of  $G$  that is a tree. The *tree edges* in  $G$  are the edges in  $G$  that belong to a given spanning tree; all other edges in  $G$  are *nontree edges*. If each edge in  $G$  has a real *weight*, the weight of a spanning tree of  $G$  is the sum of the weights of its edges. A *minimum spanning tree* in  $G$  (with respect to these weights) is a spanning tree in  $G$  with minimum weight.

Let  $G = (V, E)$  be a graph and let  $V_1, \dots, V_k$  be disjoint nonempty subsets of vertices in  $G$ . Let  $v_1, \dots, v_k$  be  $k$  new vertices (that do not belong to  $V$ ). Define a mapping  $f$  from  $V$  to  $V \cup \{v_1, \dots, v_k\}$  by  $f(v) = v_i$  if  $v \in V_i$  ( $1 \leq i \leq k$ ) and  $f(v) = v$  if  $v$  does not belong to any  $V_i$ . The operation of *collapsing the vertices in  $V_1, \dots, V_k$*  consists in replacing the vertices in each  $V_i$  by  $v_i$ , replacing each edge  $(v, w)$  in  $G$  by the edge  $(f(v), f(w))$  and deleting all self-loops in the resulting graph. Let  $G'$  denote the resulting graph. The graph  $G'$  is a multigraph. It will often be convenient to *identify* each edge  $(v, w)$  in  $G$  where  $f(v) \neq f(w)$  with a unique edge in  $G'$  linking the vertices  $f(v)$  and  $f(w)$ . In these cases we may pick an arbitrary one-to-one mapping between the edges in  $G'$  connecting any two distinct vertices  $x$  and  $y$  and the edges  $(v, w)$  in  $G$  with  $f(v) = x$  and  $f(w) = y$  and identify an edge  $(x, y)$  with the edge in  $G$  to which it is mapped under this one-to-one mapping.

A graph is  *$k$ -edge-connected* ( $k \geq 1$ ) if the removal of at most  $k - 1$  edges does not disconnect  $G$ . An equivalent condition for a graph to be 2-edge-connected is that every edge lies on a cycle. A *2-edge-connected component* of  $G$  is a maximal 2-edge-connected subgraph of  $G$  (i.e., a 2-edge-connected component is not a proper subgraph of a 2-edge-connected subgraph of  $G$ ). A *cutedge* in  $G$  is an edge in  $G$  whose removal disconnects  $G$ . Thus, a graph is 2-edge-connected if it is connected and does not contain a cutedge.

A graph is  *$k$ -vertex-connected* ( $k \geq 1$ ) if at least  $k$  vertices have to be removed from the graph in order to disconnect it or reduce it to a single vertex. A vertex in  $G$  is a *cutpoint* if removing it together with all incident edges yields a graph that is not connected. A graph  $G$  is *biconnected* (or 2-vertex-connected) if it has at least 3 vertices and does not contain a cutpoint. A *block* (or a *biconnected component*) of a graph  $G$  is either an isolated vertex in  $G$ , or a cutedge in  $G$ , or a

## 2 Definitions

A *graph*  $G = (V, E)$  consists of a set  $V$  of *vertices* and a set  $E$  of *edges*. We sometimes write  $V(G)$  and  $E(G)$  for the set of vertices and edges, respectively, and write  $n(G)$  and  $m(G)$ , respectively, for their cardinalities. If the edges are ordered pairs  $(v, w)$  of distinct vertices, the graph is *directed*:  $v$  is called the *tail* and  $w$  is called the *head* of the edge. We also use the term *digraph* for a directed graph. The *indegree* (*outdegree*) of a vertex in a digraph is the number of edges in the digraph whose head (tail) is this node. If the edges are unordered pairs of distinct vertices, also denoted by  $(v, w)$ , the graph is *undirected*. In that case  $v$  and  $w$  are *incident* with the edge  $(v, w)$ . If  $v = w$ , then the edge  $(v, w)$  is a *self-loop*. The *degree* of a vertex is the number of edges incident with it. If  $E$  is a multiset, i.e., if edges may have multiple copies, then  $G$  is a *multigraph*.

Fix a multigraph  $G = (V, E)$ . A *path*  $P$  in  $G$  is a sequence  $\langle v_0 \dots v_k \rangle$  of vertices of  $V$  such that  $(v_{i-1}, v_i)$  ( $1 \leq i \leq k$ ) is an edge in  $E$ ; the nodes  $v_0$  and  $v_k$  are the *endpoints* of  $P$  and  $v_1 \dots v_{k-1}$  are the *internal vertices* of  $P$ . We say that  $v_k$  is *reachable* from  $v_0$ . The vertices  $v_i$  *lie on the path*  $P$  and the edges  $(v_{i-1}, v_i)$  are the *edges on the path*  $P$ . The *length* of a path is the number of edges on the path. A path is *simple* if  $v_0 \dots v_{k-1}$  are distinct and  $v_1 \dots v_k$  are distinct. A simple path is a *chain* if all of its internal vertices have degree 2 in the graph. A *cycle* in  $G$  is a path in  $G$  whose endpoints coincide and all of whose edges are distinct. A *simple cycle* is a simple path whose endpoints coincide.

If  $G = (V, E)$  and  $G' = (V', E')$  are two graphs such that  $V' \subseteq V$  and  $E' \subseteq E$ , then  $G'$  is a *subgraph* of  $G$ . A subgraph  $G'$  of  $G$  is a *proper subgraph* of  $G$  if it is different from  $G$ . A subgraph  $G'$  of  $G$  having some property is a *maximal* subgraph with this property if it is not a proper subgraph of another subgraph of  $G$  with this property.  $G'$  is the subgraph of  $G$  *induced by the vertices* in  $V'$  if  $E'$  contains exactly the edges of  $E$  between vertices of  $V'$ .  $G'$  is the subgraph of  $G$  *induced by the edges* in  $E'$  if  $V' = V$ . A *spanning subgraph* of  $G$  is a subgraph  $G'$  with  $V' = V$ . If  $G'$  is a spanning subgraph of  $G$  and  $E'' \subseteq E$ , then  $G' + E''$  denotes the graph with vertex set  $V' (= V)$  and edge set  $E' \cup E''$  and  $G' - E''$  denotes the graph with vertex set  $V'$  and edge set  $E' - E''$ .

An undirected graph  $G$  is *connected* if every vertex in  $G$  is reachable from any other vertex in  $G$ . A *connected component* in  $G$  is a maximal connected subgraph of  $G$  (i.e., it is connected and it is not a proper subgraph of a connected subgraph of  $G$ ).

A *tree* is a connected (undirected) graph without cycles. A *subtree* of a tree is a subgraph of a tree that is a tree. If  $T$  is a tree and  $u$  and  $v$  are two vertices in  $T$ , then the graph  $T + (u, v)$  contains a unique (simple) cycle called the *fundamental cycle* of  $(u, v)$  in  $T$ . A *forest* is a graph without cycles or, equivalently, it is a graph whose connected components are trees. A *leaf* in a forest is a vertex of degree at most 1 in the forest.

of  $G$ . The total time is dominated by  $m$  calls to the algorithm for testing 2-edge-connectivity ([21], [14], [13], [18]), giving a time bound of  $O(m(n+m))$ . The time can be brought down to  $O(m+n^2)$  by first finding a sparse 2-edge-connected spanning subgraph of  $G$  (see section 3). There is a similar sequential algorithm with the same time bound for finding a minimal biconnected spanning subgraph of  $G$ . None of these algorithms lends itself to an efficient parallel implementation.

In this paper we present fast parallel algorithms for these problems: they both run in time  $O(\log^3 n)$  with a number of CRCW processors close to  $(n+m)/\log n$  (for the exact bound, see end of section 3). These algorithms are the first efficient parallel algorithms for these problems. In the outer loop of both parallel algorithms we find a spanning tree in the current subgraph of the input graph that contains the smallest possible number of *redundant edges* (i.e., edges that can be removed without destroying the desired property - 2-edge-connectivity or biconnectivity). A similar step is used in each iteration of an algorithm of [7] to compute a minimal strongly connected spanning subgraph of a strongly connected digraph (*transitive compaction problem*). We augment such a spanning tree with a minimal set of edges restoring the desired property (2-edge-connectivity or biconnectivity). We use tree contraction to construct such a minimal augmentation. This part of our algorithm markedly differs from the corresponding step in the transitive compaction algorithm. As in [7] we show that  $O(\log n)$  iterations of this procedure yield the desired spanning subgraph.

We also give linear time algorithms for minimally augmenting a spanning tree into a 2-edge-connected or biconnected graph. If used in the obvious way these procedures yield sequential algorithms for both problems that run in time  $O(m+n \log n)$ . In recent work ([9]) the basic algorithms have been refined into linear time algorithms for finding a minimal 2-edge-connected spanning subgraph of a 2-edge-connected graph and finding a minimal biconnected spanning subgraph of a biconnected graph. These algorithms use the linear time minimal augmentation procedures described in this paper as subroutines. Similar linear time augmentation procedures have been found independently by Han and Tarjan ([8]).

Our paper is organized as follows. In the next section we introduce the graph-theoretic terminology. In section 3 we present the parallel algorithm for finding a minimal 2-edge-connected spanning subgraph and in section 4 we describe our parallel algorithm for finding a minimal biconnected spanning subgraph. In section 5 we present linear time sequential algorithms for minimally augmenting spanning trees with respect to these properties. In section 6 we summarize our results and mention some related results.

# On Finding Minimal Two-Connected Subgraphs \*

Pierre Kelsen

Vijaya Ramachandran

*Department of Computer Sciences*

*University of Texas, Austin, TX 78712*

July 30, 1992

## Abstract

We present efficient parallel algorithms for the problems of finding a minimal 2-edge-connected spanning subgraph of a 2-edge-connected graph and finding a minimal biconnected spanning subgraph of a biconnected graph. The parallel algorithms run in polylog time using a linear number of PRAM processors. We also give linear time sequential algorithms for minimally augmenting a spanning tree into a 2-edge-connected or biconnected graph.

## 1 Introduction

In this paper we consider the following two related problems: given a 2-edge-connected (biconnected) graph  $G$ , compute a minimal 2-edge-connected (biconnected) spanning subgraph of  $G$ , i.e., a 2-edge-connected (biconnected) subgraph in which the deletion of any edge destroys 2-edge-connectivity (biconnectivity). We present efficient parallel algorithms for these problems.

It is known that the corresponding problems of finding *minimum* spanning subgraphs with these properties are NP-hard ([6]). Thus, it is natural to study the simpler problem of computing *minimal* spanning subgraphs with respect to these properties. The problems considered here have applications in the context of network reliability.

Our interest in the parallel complexity of the above problems is prompted by the fact that both problems admit very simple sequential algorithms that seem hard to parallelize. Thus, new techniques are required to obtain an efficient parallel solution for these problems. Several other well-known problems share this property, e.g., the problem of computing a maximal independent set in a graph and the problem of computing a depth-first search tree in a graph (see [11]). To illustrate this, consider the problem of finding a minimal 2-edge-connected spanning subgraph. Let the input graph  $G$  have  $n$  vertices and  $m$  edges. We assume that  $G$  is 2-edge-connected. The obvious sequential algorithm finds a minimal 2-edge-connected spanning subgraph of  $G$  by examining the edges of  $G$  one at a time and removing an edge if the resulting graph is a 2-edge-connected spanning subgraph

---

\*Work supported in part by NSF Grant CCR-89-10707.