An Optimal Distributed Algorithm for Computing Bridge-Connected Components

PRANAY CHAUDHURI

Department of Electrical and Computer Engineering, Kuwait University, PO Box 5969, Safat, Kuwait
Email: pranay@eng.kuniv.edu.kw

In this paper I present a distributed algorithm that finds the bridge-connected components of a connected undirected graph. The algorithm uses $O(n)$ messages and $O(n)$ units of time, where n is the number of nodes in the graph. It is shown that the algorithm is optimal in communication complexity to within a constant factor.

Received March 14, 1997; revised June 28, 1997

1. INTRODUCTION

Consider a connected undirected graph $G =< N, E >$, where N is the set of n nodes, and E is the set of e edges. Without loss of generality, I assume N to be $\{1, 2, \ldots, n\}$. An edge $(i, j) \in E$ is a bridge of G if the removal of (i, j) disconnects G. Clearly, the removal of a bridge increases the number of connected components of a graph by one. The bridge-connected components problem consists of finding the maximal connected subgraphs of G such that none of the subgraphs contains a bridge. The best sequential algorithm for the bridge-finding problem, due to Tarjan [1], has time complexity $O(n + e)$. The bridge-connected components problem can also be solved in $O(n + e)$ time, initially, by applying Tarjan's algorithm on G and finally computing the connected components of G', where $G' =< N, E' >$ and $E' = E - \{(i, j)|(i, j)$ is a bridge of $G\}$. The bridge-finding problem has been extensively investigated in the context of parallel processing (see for example [2, 3, 4, 5, 6]). A parallel algorithm for the bridge-connected components problem is presented in [3]. In this paper, I present an optimal distributed algorithm for the bridge-connected components problem.

In general, starightforward adaptation of a parallel algorithm on a distributed model of computation seems to be difficult, since the distributed models are inherently asynchronous. In this case, the graph is embedded in a communication network and each processor is assigned with a control algorithm such that it allows the processor to identify the bridge-connected component to which it belongs. The output of the algorithm is available in a distributed manner in the sense that each node knows only its component number: the component number is a representative node number which is considered to be the component number for the connected component in question. The distributed algorithm presented in this paper uses $O(n)$ messages and $O(n)$ units of time.

2. COMPUTATIONAL MODEL

The computational model used here is an asynchronous network of processors with their own local memory, such that each processor of the network corresponds to a particular node of the graph under consideration. Moreover, corresponding to each edge of the graph, one bidirectional, non-interfering communication link is assumed. A communication link between a pair of node processors i and j consists of two channels: one for transmitting messages from i to j and one for transmitting messages from j to i. No common or global memory is shared by the node processors for interprocessor communication. Instead this is done by exchanging messages. It is assumed that the network is sufficiently reliable so there is no node or channel failure during transmission. I consider a very simple protocol for message communication similar to that used in [7, 8]. If a node A sends a message to a neighbour node B, then the message gets appended at the end of the input buffer of B and this process takes a finite arbitrary time (due to the transmission delay). Two or more messages arriving simultaneously at an input buffer are ordered arbitrarily and appended to the buffer. A node receives a message by removing it from the corresponding buffer.

I assume a simple message format for internode communication. A message is a triple and is expressed as:

<type; sender id; parameter>.

The parameter field of a message depends on its type field. For some messages the parameter field may consists of a vector of length n and for some messages it may be empty. The exact content of the parameter field for different types of messages will be discussed later. A similar strategy of incorporating a vector variable in message body has also been used in [9, 10, 11].

Two types of complexity measures are important for algorithms in such a computational model. One is the time

complexity and the other is the communication complexity. I assume that the standard logical operations on bit vectors are primitive operations. Further, an operation *val(B)* is assumed which returns the number of 1s in any bit vector B of length n. Also, an operation *first(B)* is assumed which returns the index of the first 1 of B. The most important assumption in this model is that each processor processes messages from its neighbours, performs local computations and sends messages to its neighbours such that no time is required for all these actions. In order to compute the time complexity in such a model, I assume that the delay between the time any message is transmitted along any edge and the time it is processed at its destination is at most one time unit. The communication or message complexity is the total number of messages (independent of types) sent during algorithm execution. This type of computational model, or similar concepts were used for solving various graph theory problems (see for example [8, 10, 11, 12, 13, 14, 15, 16, 17]).

3. THE ALGORITHM

3.1. Basis of the algorithm

The distributed algorithm for computing the bridge-connected components designed in this section consists of three phases. In the first phase, a spanning tree of the given undirected graph is obtained. In the second phase, the bridges of the graph are identified. Finally, in the third phase, the connected components of the graph are determined after logically removing the edges of the graph which are identified as the bridges in the previous phase. Although the three different phases as mentioned are highly inter-related, this modular approach makes the presentation of the algorithm more convenient.

The distributed algorithm is based on the following two fundamental lemmas, whose proofs are straightforward and hence omitted.

LEMMA 3.1. *Let* $G =< N, E >$ *be a connected undirected graph and* $T(r) =< N, E' >$ *be any spanning tree of* G *rooted at node* $r \in N$. *If* $(i, j) \in E$ *is a bridge of* G *then* $(i, j) \in E'$, *i.e. either* j *is a son of* i *or* i *is a son of* j *in* $T(r)$.

A node i is said to be a son of another node j, with respect to a spanning tree $T(r)$, if j is the immediate predecessor of i in $T(r)$; alternatively j is said to be the father of i.

LEMMA 3.2. *Let* $G =< N, E >$ *be a connected undirected graph and* $(i, j) \in E$. *Then* (i, j) *is not a bridge if and only if* (i, j) *belongs to a fundamental cycle of* G.

A set of fundamental cycles of graph G is a collection C of cycles of G with the property that any other cycle $c \notin C$ of G can be written as $c = c(1) \oplus c(2) \oplus ... \oplus c(k)$ for some subcollection of cycles $c(1), c(2), ..., c(k) \in C$.

Therefore, an edge of G is a bridge if it belongs to a spanning tree of G and iff it does not belong to any fundamental cycle of G. It is important to note that, for this purpose finding a set of fundamental cycles is not essential; instead it is sufficient to know whether a spanning tree edge belongs to any fundamental cycle or not.

3.2. Finding a spanning tree of G

Any spanning tree of G can be used as the basis in computing the bridges. However, the distributed depth-first search (DFS) algorithm of Sharma *et al.* [11], which is optimal in communication complexity, is well adapted to the present purpose. The algorithm uses a vector variable of length n in the message body and so does our distributed bridge-connected components algorithm. The goal in the present case is the same as that in [9, 10, 11], namely to achieve optimal time and message complexity by incorporating more information in the message body and allowing messages of varying length. It may be noted that the distributed DFS algorithm of Sharma *et al.* [11] had a number of errors which were subsequently corrected by Kumar *et al.* [10]. For the convenience of the reader I begin by presenting a distributed DFS algorithm. Although the algorithm is adapted from [10, 11], it has several modifications so that when the algorithm halts it provides some additional information at each node as the output. This information is required by the later phases of the computation of the bridge-connected components.

3.2.1. Messages used by the algorithm

- SEARCH. A SEARCH message from a node i to its unvisited neighbour j implies that node i will be the father of j in the DFS spanning tree of G under construction. On the other hand, if j has already been visited then it implies that i is a son of j in the concerned DFS spanning tree and that all the nodes of G which are reachable from j through i have already been visited.
- TERMINATE. A node i on receiving a TERM message from a node j, where j is the father of i, updates its ancestor set and sends the TERM message with the updated ancestor information to each of its sons. After sending the TERM message to all of its sons, node i terminates its algorithm.

3.2.2. Variables kept at node i

- NEIGHBOR_i. bit vector of length n
  [$\text{NEIGHBOR}_i(j) = 1$ implies that j is an adjacent node (neighbour) of i in $G (i \neq j)$]
- ANCESTOR_i. bit vector of length n
  [$\text{ANCESTOR}_i(j) = 1$ implies that j is an ancestor of i $(i \neq j)$]
- NONTREE_i. bit vector of length n
  [$\text{NONTREE}_i(j) = 1$ implies that (i, j) is a non-tree edge of G with respect to the given spanning tree $T(r)$ rooted at $r \in N$].
- SON_i:bit vector of length n
  [$\text{SON}_i(j) = 1$ implies that j is a son of $i (i \neq j)$]
- FATHER(i): integer
  [FATHER(i) is the father of node i in $T(r)$; it is assumed that FATHER$(r) = 0$]

Further, I assume that in addition to the above variables, a bit vector of length n denoted by VISITED is used by the algorithm which is initialized by the root. VISITED(i) = 1 indicates that node i has already been visited and knows its father in the DFS spanning tree.

The execution of the algorithm is started by a node designated as the root or the start node $r \in N$ which finally becomes the root of the DFS spanning tree produced. However, all the nodes of G act as the terminator. The distributed DFS algorithm is terminated when all the nodes of G have been terminated. It may be noted that the TERMINATE message is initiated by the root r when it receives a SEARCH message from each of its neighbours indicating that the search is complete. The root r after sending the TERMINATE message to each of its sons terminates its algorithm. The parameter field of the TERMINATE message carries the ancestor information. The main purpose of the TERMINATE message, in addition to circulating the termination information to all nodes, is to propagate the ancestor information to all the nodes such that after completion of the algorithm each node knows its ancestor set.

3.2.3. *Algorithm DISTRIBUTED_DFS*

Input. The neighbours of node i represented by the bit vector NEIGHBOR_i available at each node $i \in N$.

Output. At the termination of the algorithm, each node $i \in N$ is left with FATHER(i). Also the algorithm produces, for each node i, its set of sons, set of ancestors and set of nontree edges represented by bit vectors SON_i, ANCESTOR_i and NONTREE_i, respectively.

Algorithm for root r.
Initialization

```
begin
    foreach i ∈ N do
       begin VISITED(i) := 0;
             SON_r(i):= 0;
             ANCESTOR_r(i):= 0
       end
    VISITED(r) := 1;
    FATHER(r) := 0;
    ANCESTOR_r := ∅;
    send <SEARCH, ∅, VISITED> to r
end
```

On receiving <SEARCH, j, VISITED> message

```
begin
    NEIGHBOR_r := NEIGHBOR_r Λ ¬ VISITED;
    if NEIGHBOR_r ≠ ∅ then
       begin x := first(NEIGHBOR_r);
             SON_r(x) := 1;
             send <SEARCH, ∅, VISITED> to x
       end
    else begin   ANCESTOR(r) := 1;
                 foreach j ∈ S_r do
                    begin send <TERMINATE, r,
                    ANCESTOR_r > to j end;
                 terminate
       end
end
```

Algorithm for node $i (i \neq r)$:
Initialization

```
begin
    foreach j ∈ N do
       begin SON_i(j) := 0;
             ANCESTOR_i(j) := 0;
             CNEIGHBOR_i(j) := NEIGHBOR_i(j)
       end;
    FATHER(i) := 0;
end
```

On receiving <SEARCH, j, VISITED> message

```
begin
    if FATHER(i) = 0 then
       begin FATHER(i) := j;
             VISITED(i) := 1
       end;
    NEIGHBOR_i := NEIGHBOR_i Λ ¬ VISITED;
    if NEIGHBOR_i ≠ ∅ then
       begin x := first(NEIGHBOR_i);
             SON_i(x) := 1;
             send <SEARCH, i, VISITED> to x
       end
    else begin   NONTREE_i := CNEIGHBOR_i Λ ¬
                 (SON_i ∨ FATHER(i));
                 send <SEARCH, i, VISITED>
                 to FATHER(i)
       end
end
```

On receiving <TERMINATE, j, ANCESTOR_j > message

```
begin
    foreach k ∈ N do
       begin ANCESTOR_i(k) := ANCESTOR_j(k) end;
    ANCESTOR(i) := 1;
    foreach k ∈ S_i do
       begin send <TERMINATE, i, ANCESTOR_i >
             to k end;
    terminate
end
```

Every node in the graph except the root receives only one SEARCH message from its father and sends one SEARCH message to its father. The root receives a SEARCH message from itself and sends no SEARCH message to its father, since it has no father. No SEARCH message is sent to an already visited node. Finally, the algorithm at any node $i \in N - \{r\}$ is terminated on receiving a TERMINATE message from its father. The root initiates the first TERMINATE message. Therefore, a total of $3n$ messages are used by algorithm DISTRIBUTED_DFS. The total time is the time required to transmit the messages over the edges of the graph. Assuming that all messages are delivered in at most one unit of time, the total time needed to transmit $3n$ messages is $3n$ units. Thus, I have the following theorem.

THEOREM 3.1. *Algorithm DISTRIBUTED_DFS computes the DFS spanning tree of a connected undirected graph in $O(n)$ time and requires $O(n)$ messages.*

Algorithm DISTRIBUTED_DFS is essentially a distributed implementation of the sequential DFS algorithm and hence the correctness proof of algorithm DISTRIBUTED_DFS is omitted.

3.3. Finding the bridges of G

Each nontree edge $(i, j) \in E - E'$ when added to the spanning tree $T(r)$ creates a fundamental cycle of G. The distributed bridge finding algorithm presented in this section uses this fact and basically identifies, in a distributed manner, the set of tree edges which do not belong to any of the fundamental cycles created by the non-tree edges.

Every node $i \in N$ in the bridge finding algorithm has three basic tasks. First, it has to decide, based on the information received through its sons, whether it is the end-node of any bridge of G or not. The second task is to inform its father FATHER(i) whether (i, FATHER(i)) is an edge of any cycle of G through i and FATHER(i) or not. Finally, each node has to terminate its algorithm and this is done only on termination of the algorithms of all of its descendants. The details of the messages and the variables used by the algorithm are as follows.

3.3.1. *Messages used by the algorithm*

- CYCLE. A CYCLE message from a node $i \in N-\{r\}$ to its father, FATHER(i), informs about the fundamental cycles which include the tree edge (i, FATHER(i)). The parameter field of CYCLE message from j contains a vector CFOUND$_j$ of length n such that CFOUND$_j(k) = 1$ implies that (j, FATHER(j)) is an edge of a cycle involving the nodes j, FATHER(j), and k, and that k is an ancestor of j; otherwise CFOUND$_j(k) = 0$.
- BRIDGE. A BRIDGE message from a node i to its father, FATHER(i), implies that (i, FATHER(i)) is a bridge of G. The BRIDGE message has an empty parameter field.

3.3.2. *Variables kept at node i*

SON$_i$, ANCESTOR$_i$, NONTREE$_i$, and FATHER(i): as in Subsection 3.1

BRIDGE(i): set of nodes
[(i, j) is a bridge of G provided that $j \in$ BRIDGE(i)]

A detailed description of the algorithm is given below. Although I have written the algorithm for the root node $r \in N$ of the DFS spanning tree separately, there is no single initiator node. The execution of the algorithm starts at all nodes, i.e. all the nodes of G are initiators. However, the root node r acts as the terminator and terminates the entire algorithm after the termination of the algorithm at every other node $i \in N - \{r\}$.

3.3.3. *Algorithm DISTRIBUTED_BRIDGE_FINDING*

Input. The father, FATHER(i), and the bit vectors ANCESTOR$_i$, NONTREE$_i$ and SON$_i$ available at each node $i \in N$.

Output. At the termination of the algorithm, each node $i \in N$ is left with a set of nodes, denoted by BRIDGE(i), such that for each $j \in$ BRIDGE(i), (i, j) is a bridge of G. If there is no bridge of G incident on i, then the algorithm returns BRIDGE(i) = $\oslash$.

Algorithm for root r.

Initialization

begin BRIDGE(r) := $\oslash$;
TERMC(r) := val(SON$_r$)
end

On receiving <CYCLE, i, CFOUND$_i$ > message

begin TERMC(r) := TERMC(r) − 1;
if TERMC(r) = 0 **then** terminate
end

On receiving (BRIDGE, i, $\oslash$) message

begin BRIDGE(r) := BRIDGE(r) ∪ {i};
TERMC(r) := TERMC(r) − 1;
if TERMC(r) = 0 **then** terminate
end

Algorithm for node i ($i \neq r$).

Initialization

begin BRIDGE(i) := $\oslash$;
TERMC(i) := val(SON$_i$);
CFOUND$_i$:= NONTREE$_i$;
if TERMC$_i$ = 0 **then** /* node i is a leaf node in $T(r)$ */
begin
if val(CFOUND$_i$) = 0 **then** /* node i is not the end-node of any non-tree edge of G */
begin BRIDGE(i) := BRIDGE(i)∪{FATHER(i)};
send <BRIDGE; i; $\oslash$ > to FATHER(i);
terminate
end
else /* node i is the end-node of some non-tree edge(s) of G forming fundamental cycle(s) */
begin send <CYCLE; i; CFOUND$_i$ > to FATHER(i);
terminate
end
end
end

On receiving <CYCLE, j, CFOUND$_j$ > message

begin CFOUND$_i$:= CFOUND$_i$ ∨ CFOUND$_j$;
TERMC(i) := TERMC(i) − 1;
if TERMC(i) = 0 **then** /* all the sons of node i

have terminated their algorithms */
begin $CFOUND'_i := CFOUND_i \Lambda ANCESTOR_i$;
if val($CFOUND_i$') = 0 **then** /* none of the fundamental cycles which include node i passes through an ancestor of i, i.e., (i, FATHER(i)) is a bridge */
begin BRIDGE(i) := BRIDGE(i) $\cup$ {FATHER(i)};
send <BRIDGE; i; $\oslash$ > to FATHER(i);
terminate
end
else /* at least one fundamental cycle includes edge (i, FATHER(i)) */
begin send <CYCLE; i; $CFOUND_i$ > to FATHER(i);
terminate
end
end
end

On receiving <BRIDGE, j, $\oslash$ > message

begin BRIDGE(i) := BRIDGE(i) $\cup$ {j};
TERMC(i) := TERMC(i) - 1;
if TERMC(i) = 0 **then** /* all the sons of node i have terminated their algorithms */
begin $CFOUND'_i := CFOUND_i \Lambda ANCESTOR_i$;
if val($CFOUND'_i$) = 0 **then** /* (i, FATHER(i)) is a bridge */
begin BRIDGE(i) := BRIDGE(i)$\cup$ {FATHER(i)};
send <BRIDGE; i; $\oslash$ > to FATHER(i);
terminate
end
end
else /* at least one fundamental cycle includes edge (i, FATHER(i)) */
begin send <CYCLE; i; $CFOUND_i$ > to FATHER(i);
terminate
end
end

The algorithm works as follows. After initializing, every leaf node $l \in N$ which is not the end-node of any non-tree edge of G with respect to $T(r)$, identifies (l, FATHER(l)) as a bridge and sends this information to FATHER(l) using a BRIDGE message with parameter = $\oslash$. On the other hand, if a leaf node $l \in N$ finds that it is an end-node of one or more non-tree edges, then it informs its father about these cycles using a CYCLE message with parameter = $CFOUND_l$.

Every internal node $j \in N$ on receiving CYCLE and/or BRIDGE messages from each of its sons checks whether any of the fundamental cycles reported to it by its sons also passes through FATHER(j). If this test is positive then, node j sends a CYCLE message to FATHER(j) with $CFOUND_j$ as the parameter; otherwise j records (j, FATHER(j)) as a bridge and sends a BRIDGE message to FATHER(j) informing about this bridge. When a node $j \in N$ receives a BRIDGE message from a son k, it updates the bridge set BRIDGE(i) by including k in it. Each node $i \in N - \{r\}$ terminates its algorithm after sending either a BRIDGE or a CYCLE message to FATHER(i), depending on whether (i, FATHER(i)) is a bridge or not. Finally, the root $r \in N$ terminates the algorithm DISTRIBUTED_BRIDGE_FINDING after receiving CYCLE and/or BRIDGE messages from all its sons and hence updating the BRIDGE(r) set.

Assuming that a message takes at most one time unit to travel from a node to its neighbour, the total time required by algorithm DISTRIBUTED_BRIDGE_FINDING is equal to the height of $T(r)$, i.e. the largest distance between the root and a leaf node. Therefore, the time complexity of algorithm DISTRIBUTED_BRIDGE_FINDING is bounded from above by $n - 1$, i.e. $O(n)$.

Since every edge of $T(r)$ carries either a CYCLE or a BRIDGE message but not both, algorithm DISTRIBUTED_BRIDGE_FINDING requires exactly $n - 1$ messages. This implies that the message complexity of algorithm DISTRIBUTED_BRIDGE_FINDING is also $O(n)$. From this discussion, I have the following theorem.

THEOREM 3.2. *Algorithm DISTRIBUTED_BRIDGE_FINDING for determining all bridges of a connected undirected graph runs in $O(n)$ time and uses $O(n)$ messages.*

The correctness of algorithm DISTRIBUTED_BRIDGE_FINDING can be established through the following lemmas.

LEMMA 3.3. *Every node $i \in N$ receives a CYCLE (BRIDGE) message within a finite time from a node $j \in N$, such that FATHER(j) = i, if and only if edge (i, j) belongs (does not belong) to a fundamental cycle.*

Proof. According to algorithm DISTRIBUTED_BRIDGE_FINDING, a leaf node $l \in N$ in $T(r)$ initiates a CYCLE message if i is the end-node of at least one fundamental cycle. The leaf node l sends this CYCLE message with parameter = $CFOUND_l$ to its father FATHER(l). Each internal node $i \in N$ on receiving a CYCLE message from one of its sons $j \in N$ updates $CFOUND_i$ to ($CFOUND_i$ OR $CFOUND_j$). Finally, node i on receiving CYCLE/BRIDGE messages from all of its val(SON_i) sons computes $CFOUND'_i$= ($CFOUND_i \Lambda$ $ANCESTOR_i$). val($CFOUND'_i$) = 0 implies that (i, FATHER(i)) does not belong to any fundamental cycle and hence node i sends a BRIDGE message to FATHER(i) after updating BRIDGE(i) by including FATHER(i). Otherwise, node i sends a CYCLE message to FATHER(i) with $CFOUND_i$ as the parameter, since val($CFOUND'_i$) $\neq$ 0 implies that at least one fundamental cycle passes through edge (i, FATHER(i)). Since every leaf node $l \in N$ initiates the algorithm by sending either a CYCLE or a BRIDGE message

to its father and the network is assumed to be sufficiently reliable with finite transmission delay, every internal node $i \in N$ which is the father of one or more leaf nodes receives CYCLE and/or BRIDGE messages from all of its sons within a finite time. With this as the basis I now complete the proof by induction.

Assume that an internal node $i \in N$ has received CYCLE/BRIDGE messages from all of its val(SON$_i$) sons. Then, according to algorithm DISTRIBUTED_BRIDGE_FINDING, node i sends either a CYCLE or a BRIDGE message to FATHER(i) after some local computations. All these local computations take no time according to the assumption made in the computational model. Therefore, FATHER(i) must receive a CYCLE message from node i if (i, FATHER(i)) belongs to at least one fundamental cycle of G; otherwise FATHER(i) receives a BRIDGE message from node i within a finite time.

I have seen that the lemma holds for every leaf node (a leaf node has no son and hence there is no question of any CYCLE or BRIDGE message to a leaf) and every node which is a father of one or more leaf nodes. Also I have shown above that if the lemma holds for any internal node $i \in N$ then it also holds for its father FATHER(i). This completes the proof of the lemma. □

LEMMA 3.4. *Algorithm DISTRIBUTED_BRIDGE_FINDING eventually terminates.*

Proof. Algorithm DISTRIBUTED_BRIDGE_FINDING is terminated by the root $r \in N$ of $T(r)$ on receiving CYCLE and/or BRIDGE messages from all of its val(SON$_r$) sons. Every node $i \in N$–{r} terminates its algorithm after sending either a CYCLE or a BRIDGE message to its father FATHER(i). Each non-leaf node sends a CYCLE or a BRIDGE message to its father on receiving CYCLE/BRIDGE message from each of its sons.

All the leaf nodes terminate their algorithm first, since they do not require to wait for receiving CYCLE and/or BRIDGE messages from its sons. Therefore, every node $i \in N$ which is the father of one or more leaf nodes receives CYCLE/BRIDGE message from each of its sons within a finite time and terminates its algorithm after necessary updating steps and finally by sending a CYCLE or a BRIDGE message to its father FATHER(i).

Therefore, by induction it follows that within a finite time the root $r \in N$ will receive CYCLE/BRIDGE message from each of its sons and terminate the algorithm. □

LEMMA 3.5. *Every node $i \in N$ is eventually assigned with a set of nodes (possibly empty) such that each member of this set together with node i forms a bridge of G.*

Proof. Algorithm DISTRIBUTED_BRIDGE_FINDING initializes the set BRIDGE(i) to $\oslash$, for each node $i \in N$. A leaf node $l \in N$ sets BRIDGE(l) to {FATHER(l)} if and only if node l is not the end-node of any non-tree edge of G in which case edge (l, FATHER(l)) is a bridge. Every internal node $i \in N$ on receiving a BRIDGE message from node $j \in N$, such that FATHER(j) $= i$, includes node j in BRIDGE(i). If node $i \in N - \{r\}$, on receiving CYCLE/BRIDGE message from all of its sons, finds that none of the fundamental cycles reported by its sons includes an ancestor of i and also i is not the end-node of any non-tree edge of G, then it includes FATHER(i) in BRIDGE(i) and sends a BRIDGE message to FATHER(i); otherwise node i sends a CYCLE message to FATHER(i) with appropriate fundamental cycles information. By Lemma 3.3, every node i receives CYCLE/BRIDGE message from each of its sons within a finite time. Therefore, within a finite time every node $i \in N$ is assigned with a set of nodes BRIDGE(i) (possibly empty), such that for each $j \in$ BRIDGE(i), (i, j) is a bridge of G. □

From the above lemmas, the theorem stated below immediately follows.

THEOREM 3.3. *Algorithm DISTRIBUTED_BRIDGE_FINDING correctly computes the set of bridges of a connected undirected graph G in a distributed manner within a finite time.*

3.4. Finding the bridge-connected components of G

The bridge-connected components (BCCs) of an undirected graph, which is the third phase of our computation, is performed as follows. Once the DFS spanning tree and the bridges of G are available, all the bridges of G are logically removed from the DFS spanning tree, $T(r)$, leaving a DFS spanning forest. All nodes corresponding to each tree in the DFS spanning forest constitutes a BCC. The node number of the root for each tree in the DFS spanning forest is considered as the BCC number for each node in the concerned tree. This is a relatively simple computation in comparison with the previous two phases where the DFS spanning tree and the bridges of G are computed. Only a single type of message is used during this phase. The details of the message and the variables used by the algorithm are given below.

3.4.1. Message used by the algorithm

- COMPONENT. A COMPONENT message to a node i carries the BCC number of the sender node as the parameter which also becomes the BCC number of node i in G. Node i on receiving the COMPONENT message sets its BCC number as the BCC number of the sender node. Then node i sends COMPONENT message to each node $j \in$ SON$_i$–BRIDGE(i) and terminates its algorithm.

3.4.2. Variables kept at node i

- SON$_i$, BRIDGE(i) and FATHER(i): as in Subsection 3.2
- BCC(i):integer [BCC(i) is the bridge-connected component number of node i in G]

A detailed description of the algorithm is provided below. It may be noted that in the present case there is no single initiator node. The execution of the algorithm starts at all

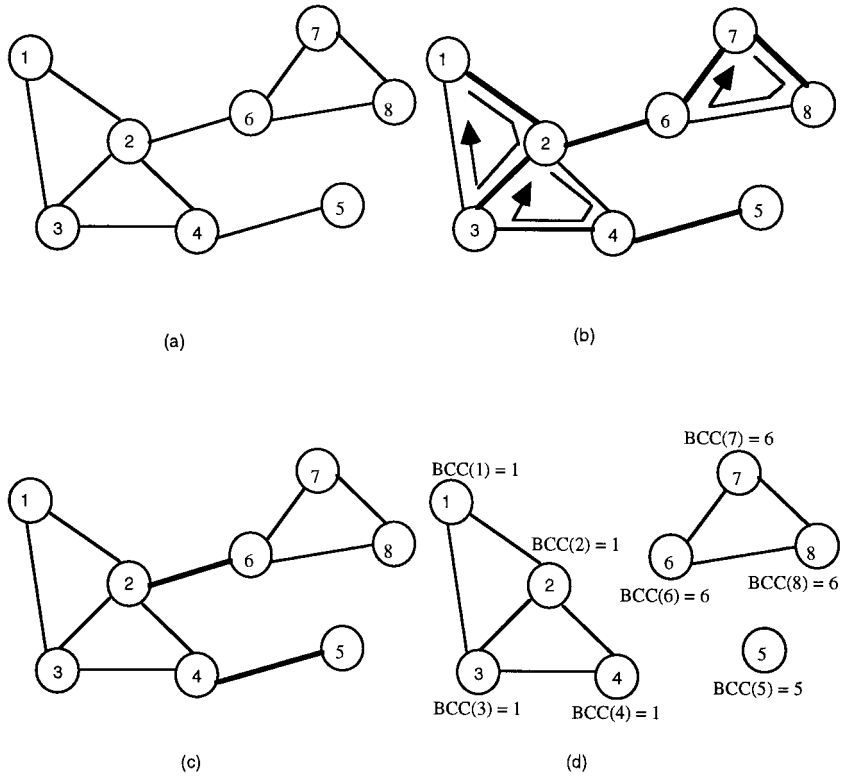

FIGURE 1. **(a)** An arbitrary connected undirected graph G. **(b)** The DFS spanning tree of G rooted at node 1. The tree edges are shown by bold lines. Also the fundamental cycles of G with respect to the DFS spanning tree rooted at node 1 are shown. **(c)** The bridges of G are shown by bold lines. **(d)** The bridge-connected components of G. Each node is assigned with the bridge-connected component number to which it belongs.

nodes, i.e. all the nodes of G act as the initiator. Moreover, the same is true during the termination of the algorithm, i.e. all the nodes of G act as the terminator of the algorithm.

3.4.3. Algorithm DISTRIBUTED_BCC

Input. The father denoted by FATHER(i), the set of nodes BRIDGE(i), and the bit vector SON_i available at each node $i \in N$.

Output. At the termination of the algorithm, each node $i \in N$ is left with an integer BCC(i) which represents the bridge-connected component to which node i belongs.

Algorithm for node i.

Initialization

```
begin   SON'_i := SON_i–BRIDGE(i);
        if FATHER(i) ∈ BRIDGE(i) then
            begin FATHER(i) = 0 end;
        if FATHER(i) = 0 then
            begin   BCC(i) := i;
                    foreach j ∈ SON'_i do
                        begin send (COMPONENT, i,
                        BCC(i)) to j end;
                    terminate
            end
end
```

On receiving <COMPONENT, j, BCC(j) > message

```
begin   BCC(i) := BCC(j);
        foreach j ∈ SON'_i do
            begin send (COMPONENT, i, BCC(i)) to j
            end;
        terminate
end
```

The algorithm works as follows. During initialization all the bridges of G are logically removed in a distributed manner. Then, the root $i \in N$ of each tree in the DFS forest so generated initiates a COMPONENT message and sends to each of its remaining sons after deletion of those sons which creates bridges with node i. After this node i terminates its algorithm. Any node $j \in N$ which is not a root of any tree in the DFS forest on receiving a COMPONENT message sets its BCC number and sends COMPONENT message to each of its remaining sons as mentioned before, and then terminates its algorithm. Clearly, a maximum of n COMPONENT messages can be exchanged through the network during the entire computation. Thus, I have the following theorem.

THEOREM 3.4. *Algorithm DISTRIBUTED_BCC for determining all bridges of a connected undirected graph runs*

in $O(n)$ time and uses $O(n)$ messages.

The various phases of the computation of the bridge-connected components are illustrated in Figure 1 with reference to an arbitrary connected undirected graph.

The correctness of algorithm DISTRIBUTED_BCC can be established through a similar set of lemmas as used in Subsection 3.3, and following the correctness of algorithm DISTRIBUTED_DFS and DISTRIBUTED_BRIDGE_FINDING. Finally, the optimality of algorithm DISTRIBUTED_BCC is established in the following theorem.

THEOREM 3.5. *Algorithm DISTRIBUTED_BCC is optimal in communication complexity.*

Proof. Any algorithm that solves the bridge-connected components problem must examine every edge of G at least once. In a distributed computational environment at least one message is required to examine a single edge of G. Clearly, there cannot exist a distributed BCC algorithm which requires less than n messages. Therefore, $\Omega(n)$ is the lower bound to the communication complexity for finding all the BCCs of a connected undirected graph on a distributed model of computation. The communication complexity of algorithm DISTRIBUTED_BCC, presented in this section, is exactly identical with this lower bound, and hence algorithm DISTRIBUTED_BCC is optimal in communication complexity to within a constant factor. □

4. CONCLUSION

In a distributed or network model of computation, I have presented an algorithm that identifies the bridge-connected components of a connected undirected graph. Regarding the input, it is assumed that every node knows about its neighbour node numbers. Starting only with the neighbourhood information of each node of G, I have shown that it is possible to find the bridge-connected components of G in $O(n)$ time using $O(n)$ messages. The proposed algorithm is optimal in communication complexity.

ACKNOWLEDGEMENTS

I would like to thank the anonymous referees for detailed and helpful suggestions on an earlier version of this paper. These suggestions have greatly enhanced the readability of the paper. This research is supported in part by Kuwait University Grant EE065.

REFERENCES

[1] Tarjan, R. E. (1974) A note on finding the bridges of a graph. *Info. Process. Lett.*, **2**, 160–161.

[2] Chaudhuri, P. (1992) *Parallel Algorithms: Design and Analysis*. Prentice-Hall, Sydney.

[3] Ghosh, R. K. (1986) Parallel algorithms for connectivity problems in graph theory. *Int. Comput. Math.*, **18**, 193—218.

[4] Savage, C. and Ja' Ja', J. (1981) Fast, efficient parallel algorithms for some graph problems. *SIAM J. Comput.*, **10**, 682–691.

[5] Tarjan, R. E. and Vishkin, U. (1983) *An efficient parallel biconnectivity algorithm*. TR-69, Ultracomputer Note-51, Courant Institute of Mathematical Sciences, NY.

[6] Tsin, Y. H. and Chin, F. Y. (1984) Efficient parallel algorithms for class of graph theoretic problems. *SIAM J. Comput.*, **13**, 580–599.

[7] Dijkstra, E. W. and Scholten, C. S. (1980) Termination detection for diffusing computation. *Info. Process. Lett.*, **11**, 1–4.

[8] Misra, J. and Chandy, K. M. (1982) A distributed graph algorithm: knot detection. *ACM Trans. Prog. Lang. Syst.*, **4**, 678–686.

[9] Finn, S. G. (1979) Resynch procedures and a fail-safe network protocol. *IEEE Trans. Soft. Engng*, **SE-27**, 840–845.

[10] Kumar, D., Iyenger, S. S. and Sharma, M. B. (1990) Corrections to a distributed depth-first search algorithm. *Info. Process. Lett.*, **35**, 55–56.

[11] Sharma, M. B., Iyenger, S. S. and Mandyam, N. K. (1989) An efficient distributed depth-first search algorithm. *Info. Process. Lett.*, **32**, 183–186.

[12] Awerbuch, B. (1985) A new distributed depth-first search algorithm. *Info. Process. Lett.*, **20**, 147–150.

[13] Chang, E. J. (1982) Echo algorithms: depth parallel operations on general graphs. *IEEE Trans. Soft. Engng*, **SE-8**, 391–401.

[14] Chaudhuri, P. (1987) Algorithms for some graph problems on a distributed computational model. *Info. Sci.*, **43**, 205–228.

[15] Chaudhuri, P. (1992) Distributed processing of graphs: fundamental cycles algorithm. *Info. Sci.*, **60**, 41–50.

[16] Cheung, T. Y. (1983) Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Trans. Soft. Engng*, **SE-9**, 504–512.

[17] Korach, E., Rotem, D. and Santoro, N. (1984) Distributed algorithms for finding centers and medians in networks. *ACM Trans. Progr. Lang. Syst.*, **5**, 380–401.