Information Processing Letters 20 (1985) 147–150
North-Holland

8 April 1985

A NEW DISTRIBUTED DEPTH-FIRST-SEARCH ALGORITHM

Baruch AWERBUCH *

IBM — Israel Scientific Center, Haifa 32000, Israel

Communicated by H.R. Wiehle
Received 20 January 1984
Revised 22 June 1984

This paper presents a new distributed Depth-First-Search (DFS) algorithm for an asynchronous communication network, whose communication and time complexities are O(|E|) and O(|V|), respectively. The output of the algorithm is the DFS tree, kept in a distributed fashion. The existing algorithm, due to Cheung (1983), requires O(|E|) both in communication and time complexities.

Keywords: Distributed system, communication graph

1. The model

An asynchronous network is a point-to-point (or store-and-forward) communication network, described by an undirected *communication graph* (V, E) where the set of nodes V represents processors of the network and the set of edges E represents bidirectional non-interfering communication channels operating between them (see also [1,2]). No common memory is shared by the node's processors. Each node processes messages received from its neighbors, performs local computations, and sends messages to its neighbors. All these actions are assumed to be performed in *zero time*. The only assumption about the communication subsystem is that each message sent by a node to its neighbor arrives to it within some finite undetermined time. Unlike most works dealing with the same model, we do not require that the messages sent over the same link obey the *FIFO* rule.

The following complexity measures are used to evaluate performances of distributed algorithms operating in the above network. The *communication complexity*, C, is the total amount of messages sent during the algorithm. The *time complexity*, T, is the time passed from its starting time to its termination, assuming that delay of an edge is (at most) one time unit. This is under the provision that the algorithm operates correctly without this assumption.

2. The problem and the existing solutions

In this paper we present a new distributed Depth-First-Search (DFS) algorithm for the network described above. The output of such an algorithm is a DFS tree [5] of the communication graph (V, E), kept in a distributed fashion, i.e., each node must know its father in the tree. Our algorithm can be easily extended to detect the biconnected components or the strongly-connected components (in the directed case).

These two tasks have important applications in computer networks. Strongly-connected components can be used in order to detect directed cycles in communication paths, which may lead to deadlocks. Biconnected components may be used to check whether failure of one node may or may not disconnect the network.

The best existing DFS algorithm in the sense as

* Present affiliation: Computer Science Laboratory, MIT, Cambridge, MA 02139, U.S.A.

described above was given in [1]. The algorithm of [2] only finds the biconnected components, *without* constructing the DFS tree. In [3,4] a different computation model was considered, where all the processors share a common memory; thus these results cannot be applied in our case.

In the implementation of [1], movement of the center of activity of the search [5] (referred to in future as the 'center') over a tree edge or scanning of a back edge is represented by passage of a certain message over the corresponding network edge. Suppose that such a message arrives at node j from node i. If node j was not yet visited [(i − j) is a *tree edge*], then the center moves to node j, and j marks node i as its *father*. Otherwise (j was visited and thus (i − j) is a *back edge*), the above message is returned to i, and i learns that j was visited. Whenever the message was returned from each of the neighbors of the center (all neighbors are known as *visited*) the message is returned back to the father. Since each edge of the graph is traversed exactly once in each direction, the communication complexity of this algorithm is $C = O(|E|)$ messages. Since these traversals are *serial*, i.e., performed one-by-one, the time complexity is $T = O(|E|)$ time units.

3. Our solution

The major reason for the high time cost of the above algorithm is the fact that all the edges are traversed *serially*. The idea of our algorithm is to perform traversals along back edges in *paralllel*, so that only tree edges will be traversed serially. This will reduce the time complexity from $O(|E|)$ to $O(|V|)$, without changing the communication complexity.

The algorithm operates as follows. Whenever a node receives a *DISCOVER* message, it means that this node is visited for the *first time*, and the sender of the message is its father in the tree. At that time, the node sends (simultaneously) a special message *VISITED* to all the neighbors, except for the father. Now, the search process is temporarily suspended. Upon receipt of the *VISITED* message, a node learns that the sender of the message was visited, and then sends back an acknowledgement *ACK*. Whenever the *ACK* messages have been collected from all the neighbours, the center delivers a *RETURN* message to itself. Whenever a node receives the *RETURN* message, it means that the search is resumed from that node. Namely, if there exists an unvisited neighbor, the *DISCOVER* message is forwarded to it, and it is marked as *visited*; otherwise, the *RETURN* message is sent back to the father. The formal presentation of this algorithm is given in Appendix A.

It can be easily shown that at the time when the center arrives at a certain node, it knows exactly which of its neighbors were visited until now. Thus, *DISCOVER* is never sent over back edges, unlike the algorithm of [1]. This is the reason for the saving in time. Observe that exactly two messages, *DISCOVER* and *RETURN*, are sent over each tree edge in opposite directions. Also, one *VISITED* message is sent by each node over all incident edges, except for the one leading to the father, and each such message is acknowledged by the *ACK* message. Thus, the total communication complexity of the algorithm is $C = 4|E| = O(|E|)$. Observe that only the tree edges are traversed serially, and the additional delay of (at most) two time units is introduced at each node with degree greater than one at the time when it is discovered (this is the time needed to send the *VISITED* messages to all neighbors and to receive the *ACK* messages back). Thus, the total time complexity of the algorithm is $T \leqslant 4|V| - 2 = O(|V|)$.

Acknowledgment

The author wishes to thank Dr. Shmuel Katz who suggested the author the final form of the algorithm and who made a number of helpful comments.

Appendix A. The formal presentation of the algorithm

Here, we give a formal algorithm performed by each node i of the network. The algorithm specifies the actions taken by node i in response to messages arriving to it from its neighbors or from itself. Say, "**for** *DISCOVER* **from** q **do**..." means: "After receipt of *DISCOVER* message from neighbor q, perform..."

Messages of the algorithm

DISCOVER = message arriving at a node when it is visited for the first time,
RETURN = message returning the center to a node which was already visited in the past,
VISITED = sent by a node whenever it is visited for the first time,
ACK = acknowledgement, sent in response to the *VISITED* message.

Variables kept at node i

Neighbors(i) = the set of neighbors of node i (input to the algorithm),
Father(i) = the father of i in the DFS tree (output of the algorithm). At the start-point s of the search, *Father*(i) = i,
Unvisited(i) = the subset of *Neighbors*(i) including the neighbors from which the *VISITED* message was not yet received. Initially, *Unvisited*(i) = *Neighbors*(i),
flag(i, j) = a binary flag, kept for each j ∈ *Neigbors*(i), which is equal to 1 in the interval after *VISITED* was sent from i to j and before *ACK* was received back. Initially, *flag*(i, j) = 0.

Initialization of the algorithm:
A node s is chosen as a start-point of the search. To trigger the algorithm, it delivers a *DISCOVER* message to itself.

```
The algorithm for node i
  for DISCOVER message from j do  /* i is visited for the first time */
  Father(i) ← j
  for all q ∈ Neighbors(i) except for node j do
    send VISITED message to q
    flag(i, q) ← 1
  end
  if Neighbors(i) = j then send RETURN to j   /* j is the only neighbor of i */
end

for RETURN message from q do
/* the search is resumed from node i which was already visited in the past */
  if there exists k ∈ Unvisited(i) then do
    send DISCOVER to k
    remove k from Unvisited(i)
  end
  else /* all the neighbors were visited */
       if Father(i) ≠ i then send RETURN to Father(i)  /* backtracking */
       else STOP, the algorithm has terminated
end
```

```
for VISITED message from k do
  Drop k from Unvisited(i)
  send ACK message to k
end

for ACK message from j do
    flag(i, j) ← 0
    if flag(i, q) = 0 for all q ∈ Neighbors(i) then deliver RETURN to itself
    /* continue the search, having collected all the acknowledgements */
end
```

References

[1] T. Cheung, Graph traversal techniques and the maximum flow problem in distributed computation, IEEE Trans. Software Engineering SE-9 (4) (1983) 504–512.

[2] E.J.H. Chang, Echo algorithms: Depth parallel operations on general graphs, IEEE Trans. Software Engineering SE-8 (4) (1982) 391–401.

[3] E.R. Arjomandi and D. Corneil, Parallel computations in graph theory, SIAM J. Comput. 7 (1978).

[4] D. Eckshtein and D. Alton, Parallel graph processing using depth-first search, Proc. Conf. on Theoretical Computer Science, Univ. Waterloo, Canada, 1977.

[5] S. Even, Graph Algorithms (Computer Science Press, Potomac, MD, 1979).