Information Processing Letters 26 (1987/88) 301–305
North-Holland

25 January 198

YET ANOTHER DISTRIBUTED DEPTH-FIRST-SEARCH ALGORITHM

Isreal CIDON

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.

Communicated by David Gries
Received 24 October 1986
Revised 15 January 1987 and 16 June 1987

A new distributed depth-first-search algorithm is presented whose communication and time complexities are bounded by $3|E|$ and $2|V|$, respectively.

Keywords: Distributed system, distributed algorithm, asynchronous, depth-first-search

1. Introduction

In a recent paper [1], Awerbuch suggests a distributed depth-first-search (DDFS) algorithm that improves the time complexity of the previous DDFS algorithm of Cheung [2]. The communication cost of the algorithm of [1] is $4|E|$ messages and the time complexity is less than $4|V|$, where E is the set of undirected links and V is the set of nodes of the communication network. This is compared to $2|E|$ for both complexities in the previous algorithm of [2].

This paper presents a DDFS algorithm that uses no more than $3|E|$ messages and $2|V|$ units of time, thus improving both the communication and the time complexities of [1]. In addition, as in [1], no FIFO rule of message delivery is needed for the operation of the DDFS algorithm.

2. The model

Let G(V, E) represent the communication network, with V being the set of nodes and E the set of bidirectional communication links. The link between nodes i and j is denoted by the unordered pair (i, j) and carries messages in either directions. The asynchronous network has the following properties:

(1) All messages sent from i to j over link (i, j) arrive correctly, within arbitrary but finite time, and not necessarily in the same order they were sent.

(2) Each node is aware of all its links. It knows the identity of the link over which a message is received.

The following complexity measures are used to evaluate performances of distributed algorithms operating in the above network. The *communication complexity* is the total number of messages sent during execution of the algorithm. The *time complexity* is the maximum time passed from its start to its termination, assuming that the time of delivering a message over each link is at most one unit of time. This bounded delay is assumed only for evaluating the time complexity. The algorithm operates correctly with any finite arbitrary message-delivery time.

3. The DDFS algorithm

We present a new DDFS algorithm for the network described above. The output of such an algorithm is a DFS tree of the graph G(V, E) kept in a distributed fashion, i.e., each node knows its links to its father and sons in the tree. A DFS tree is defined by the centralized DFS algorithm, formally described in [3].

3.1. Informal description of the DDFS algorithm

As in all previous DDFS algorithms, a token is passed sequentially from node to node in order to explore the network. The key idea that accelerates the completion time of the algorithm lies in the fact that each node, upon receiving the token for the first time, notifies its neighbors that it has been visited. This will prevent them from considering this node as unexplored and from sending the token to it at some later time. The same idea is used in [1].

In [1], an acknowledgement is sent for each notification and the node holds the token until all notifications are acknowledged. In our algorithm, no acknowledgements are used, so no time is spent on waiting for them. The token is forwarded immediately to the next node when the notifications are sent. Thus, in contrast to [1], the token may be sent to an already explored node (whose notification has not yet been received). In such a case, the later arrival of the notification indicates to the sender that it has sent the token to an already explored node and that it was rejected. The sender generates the token anew and proceeds to explore the other neighbors. Even though the token may be sent unnecessarily to some nodes, both communication and time costs are improved.

We now give a more detailed description of the actual steps taken by the nodes.

At each node, all links may have one of the following four possible markings: *unvisited* (initial marking), *visited*, *son*, and *father*. The node itself may be in one of the two states *idle* (initial state) and *discovered*. Execution begins with a unique source node transiting from the *idle* to the *discovered* state, selecting one of the *unvisited* links, marking it as *son*, and sending the message TOKEN over that link. In addition, the node sends VISITED messages over all other links.

Upon receiving the TOKEN message for the first time over a link, a node transits to the *discovered* state and marks the link as *father*. If it has some *unvisited* links, the node selects one of them, sends the TOKEN over that link, marks it as *son*, and broadcasts a VISITED message over all *unvisited* and *visited* links (excluding the selected link, which is now marked as *son*). If no *unvisited* links exist, the node sends the TOKEN over its *father* link. If no *father* link exists, the algorithm terminates (since only the source may have no *father* link).

If the TOKEN message is received over a *son* link, the node does the same as above (excluding the broadcast of the VISITED messages).

Upon receiving the TOKEN over an *unvisited* or *visited* link but not for the first time, no response is sent. The algorithm relies on the fact that a VISITED message was sent in the past over that link. The node just marks that link as *visited* if not already done so.

Upon receiving a VISITED message over an *unvisited* link, the node marks that link as *visited*. If it receives the VISITED message over a link marked as *son*, the node marks it as *visited*. This reception implies that the TOKEN was previously sent over that link and was rejected by its neighbor. At this point, the node acts as it would act if it received the TOKEN back from one of its sons (as described above). In all other cases, the VISITED message is ignored. (If it was received over a *father* link, then the node realizes that the message was received out of order since it is always sent prior to the TOKEN message. Consequently, there is no meaning to this message and it is ignored.)

3.2. Formal description of the DDFS algorithm

In the following, we give the algorithm executed by each node i. Variable $mark_i(j)$ contains the current marking of link j at node i, and variable $state_i$ describes the state of node i.

The following messages and signals are used

START: a signal sent from the outside world in order to start the algorithm,
TOKEN: a message sent to a neighbor perceived to be in the idle state,
VISITED: a message sent in order to notify neighbors that the node has already transited from the idle state.

ROUTINE: MAIN

On receiving START from outside and i = source:
if $STATE_i = IDLE$ **then**
$STATE_i \leftarrow discovered$;
SEARCH;
send VISITED over all k such that $mark_i(k)$ is *visited* or *unvisited*

fi.

On receiving TOKEN over link j:
if $STATE_i = IDLE$ **then**
$mark_i(j) \leftarrow father$;
$STATE_i \leftarrow discovered$;
SEARCH;
send VISITED over all k such that $mark_i(k)$ is *visited* or *unvisited*
else
if $mark_i(j) = unvisited$ **then** $mark_i(j) \leftarrow visited$ **fi**
if $mark_i(j) = son$ **then** SEARCH **fi**

fi.

On receiving VISITED over link j:
if $mark_i(j) = unvisited$ **then** $mark_i(j) \leftarrow visited$ **fi**
if $mark_i(j) = son$ **then**
$mark_i(j) \leftarrow visited$;
SEARCH

fi.

ROUTINE: SEARCH
if for some k $mark_i(k) = unvisited$ **then**
send TOKEN to k;
$mark_i(k) \leftarrow son$
else **if** i = souce **then** stop **else** send TOKEN over k for which $mark_i(k) = father$ **fi**
fi.

---

4. Proof of correctness and complexity bounds

The above DDFS algorithm visits all nodes in the network and terminates at the source node. It also distributively constructs a DFS tree routed at the source node, with each node knowing its father and sons in the tree. Below we prove this and the communication and time complexities bounds.

4.1. Lemma. *If all messages are delivered in at most one unit of time, then the algorithm terminates after at most* $2|V|-2$ *units of time.*

Proof. First we prove that if the TOKEN is sent to an *idle* node i by its father at time t, then, in at most one unit of time after t, the TOKEN is sent to another *idle* node or back to the father.

All VISITED messages destined for i from nodes in the *discovered* state were sent prior to the father's TOKEN. Consequently, they all have been received at i by time $t+1$. At that time, node i is aware if it has sent the TOKEN to some *discovered* node over a link marked as *unvisited*. At this point, i sends the TOKEN over an *unvisited* link, which always leads to an *idle* node (if one exists), or back to its father. In the latter case, the TOKEN is never forwarded again to i by its father.

Since the TOKEN is transmitted once in each direction over all links of the DFS tree and the time between two consecutive transmissions is bounded by one, the total time spent by the DDFS is bounded by $2|V|-2$. □

Note that if all messages are delivered in exactly one unit of time (the network is synchronous), the algorithm terminates after exactly $2|V|-2$ units of time. Moreover, in such a case, since message delay is exactly one unit of time, whenever node i forwards the TOKEN to one of its neighbors it will have received all VISITED messages that were sent to it over all links. This implies that, in this case, the TOKEN is never sent to a *discovered* node.

4.2. Lemma. *No more than three messages are sent over any link in both directions.*

Proof. It is clear that the VISITED or TOKEN messages can be sent at most once over each link in either direction. This implies that no more than four messages are sent over any link in both directions. To prove that at most three messages are sent it is enough to show that if two VISITED messages are sent, then at most one TOKEN message is sent (since at most two messages of each type may be sent, this also implies that if two TOKEN messages are sent, at most one VISITED is sent).

Consider the case that two VISITED messages are sent over link (i, j). Also assume that a TOKEN message was sent first from i to j (which implies that i is not the son of j). In this case, the TOKEN is sent by i before receiving the VISITED message of j, otherwise no TOKEN would be sent to j. Since j sends a VISITED message over (i, j) this cannot be the first TOKEN message received by j. In such a case, when the TOKEN sent by i is received at j, link (j, i) is marked as *visited* or *unvisited*, so no response is sent. Moreover, after the TOKEN reception link (j, i) is marked as *visited*, no future TOKEN message may be sent by j to i.

We have proved that at most three messages are sent over each link. This proves that the communication cost of the DDFS algorithm is bounded by $3|E|$ messages. □

As noted before, in a synchronous network, no TOKEN message is sent except over the links of the DFS tree. Moreover, no more than one VISITED message can be sent over each link of the DFS tree in both directions. This makes the communication cost in this special case to be at most $2|E|+|V|$.

5. Discussion

Our DDFS algorithm sends at most $3|E|$ messages and has time complexity of less than $2|V|$. An interesting feature of this algorithm is that even though it may send messages to already *discovered* nodes, it still has less communication and time complexity than the algorithm of [1], which prevents such undesirable transmissions. This may suggest that the time and messages spent in recovering from unnecessary actions is some-

times less than the time and messages spent in avoiding these actions completely.

Another interesting question arises from comparing the best message complexity DDFS of [2], which requires $2|E|$ messages, with that of this paper, which requires at most $3|E|$ messages (for some examples in a full graph, $O(3|E|)$ messages are actually sent). It seems that $2|E|$ serves as a lower bound for DDFS's message complexity. In the synchronous case, this bound is almost met. Is it possible to design a DDFS algorithm for an asynchronous network that has $O(|V|)$ time complexity and uses only $O(2|E|)$ messages?

Acknowledgment

The author wishes to thank Dr. Inder S. Gopal and Dr. Shmuel Zaks for helpful discussions and comments. He also thanks the anonymous referees and the communicating editor for their assistance in improving the presentation of this paper.

References

[1] B. Awerbuch, A new distributed depth-first-search algorithm, Inform. Process. Lett. 20 (3) (1985) 147–150.

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

[3] S. Even, Graph Algorithms (Computer Science Press, Rockville, MD, 1979) 53–57.