Information Processing Letters 32 (1989) 183–186
North-Holland

1 September 1989

AN EFFICIENT DISTRIBUTED DEPTH-FIRST-SEARCH ALGORITHM

Mohan B. SHARMA, Sitharama S. IYENGAR

Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, U.S.A.

Narasimha K. MANDYAM

Indian Telephone Industries Ltd., Bangalore, India

Communicated by David Gries
Received 18 July 1988
Revised 11 November 1988, 26 January 1989, 30 March 1989

The sequential depth-first-search algorithm, distributed over processor nodes of a graph, yields a distributed depth-first-search algorithm that uses exactly $2|V|$ messages and $2|V|$ units of time.

Keywords: Distributed system, distributed algorithm, communication graph

1. Introduction

Consider a communication network. Our goal is to equip the set of processors in the network with a control algorithm that will allow a processor in the network to effect a depth-first traversal through the graph underlying the network, using messages. The output of the algorithm is a depth-first-search (DFS) tree of the graph underlying the network, kept in a distributed fashion, i.e., at the end of the algorithm, each node will know its parent and children in the DFS tree [5].

Table 1 contains a chronological list of distributed depth-first-search (DDFS) algorithms and their time and message complexities, considering unbounded message sizes, for an undirected graph with V nodes and E edges. The algorithm presented here achieves its optimal time [6] in a straightforward fashion. We simply distribute the traditional sequential, recursive DFS over the network, letting each processor node handle communication with its father and children. Thus, optimal time and message complexity is achieved by eliminating all real parallelism (and by putting more information in each message and allowing messages of varying size).

2. The model

The communication network is represented by the graph $G(V, E)$ where $V = \{1, 2, \ldots, \#V\}$ and E are respectively the sets of vertices and undirected edges. Vertices and edges of the graph represent the nodes and undirected communication links of the communication network. We make the following assumptions for an asynchronous communication network.

(1) No two processors in the network share memory.

(2) Any message transmitted from one node to another is received unaltered, in finite time.

(3) Each node has a distinct name (ID), and we assume that $V = \{1, 2, \ldots, \#V\}$, the set of vertices from which IDs are chosen. The total number of nodes in the network is not known a priori.

(4) Each node knows the IDs of its neighbors in the graph and the ID of the sender of each message it receives.

Table 1

Author	Year	Time complexity	Communication complexity
T. Cheung [2]	1983	$2\|E\|$	$2\|E\|$
B. Awerbuch [1]	1985	$<4\|V\|$	$4\|E\|$
K.B. Lakshmanan et al. [6]	1987	$2\|V\|-2$	$<4\|E\|-(\|V\|-1)$
I. Cidon [3]	Jan. 1988	$\leqslant 2\|V\|$	$\leqslant 3\|E\|$
this paper	July 1988	$3\|V\|$	$2\|V\|$

We evaluate the performance of the algorithm with the following complexity measurs. The time complexity is the maximum time elapsed from the beginning to the termination of the algorithm, assuming that delivering a message over a link requires at most one unit of time and that receiving a message, local processing, and sending it over a link require negligible time. However, we assume that message transmission time is small compared to the propagation delay. The communication complexity is the total number of messages sent during the execution of the algorithm.

3. Proposed DDFS algorithm

We describe the sequential DFS algorithm and its modification to function in a distributed setting. Let $M(1..\#V)$ be a Boolean array, with $M.i$ having the meaning "node i is marked". Consider the following procedure *Mark*:.

{Node i is unmarked, i.e. $M.i$ is false. Mark all unmarked nodes of G that are reachable from i along an unmarked path of nodes}
Mark: **proc** (i: integer);
  begin $M.i :=$ true;
    foreach (j: j a neighbor of i
      $\wedge \neg M.j$: *Mark*(j))
  end.

The **foreach** statement iteratively chooses (arbitrarily) an unmarked neighbor j of i and executes *Mark*(j). By induction on the length of an unmarked path from node i to node j, one can easily prove that *Mark* satisfies its specification.

Thus, if initially $M.i$ is false for all i, execution of *Mark*(root) marks all nodes reachable from node root. Procedure *Mark* uses a depth-first method of traversing G. For each reachable node i, *Mark*(i) is called exactly once, since it is called only when $M.i$ is false and the first step of the body of procedure *Mark* is to set $M.i$ to true, never to be changed again.

If a global array M is not desired, it could be made local as follows:

{Node i is unmarked, i.e. $M.i$ is false. Set $M.j$ to true for all nodes j for which $M.j$ is false and that are reachable from i along a path, all of whose nodes k have $M.k$ false}
Mark: **proc**(i: integer; **value-result** M: **array** $1..\#V$ **of** Boolean);
  begin $M.i :=$ true;
    foreach (j: j a neighbor of i
      $\wedge \neg M.j$: *Mark*(j, M))
  end.

Now consider this algorithm in a distributed system environment. Nodes and edges of the graph correspond to the nodes and the bidirectional communication links of a communication network respectively. Each node P_i will have a control algorithm similar to the body of the procedure *Mark*. A call to *Mark*(i), is replaced by a send statement *Send*(j, M) **to** j, which sends to j a message with the values j, M. Correspondingly, there is a receive statement *Receive*(k, M). Thus, the control program at each node P_i looks as follows:

P_i: **var** M: **array** $1..\#V$ **of** Boolean;
  var f: integer;
    {f will be P_i's parent}
  var s: set(int);
    {s is the set of known sons of P_i}

```
Receive(f, M);
s := { };
M.f := true;
foreach (j: j a neighbor of f ∧ ¬M.j:
        s := s ∪ {j};
        Send(j, M) to j:
        Receive(j, M));
Send(f, M)
```

The algorithm begins as follows. A node P_i begins the process of finding a DFS distributively, by either receiving a START signal from the outside world or by executing M := false; *Send*(root, M) **to** root; *Receive*(j, M), where root is the desired root of the DFS tree.

A sending node waits for node j to finish execution and to return a value M before proceeding further. Hence, the call-by-value-result parameter in procedure *Mark* is being simulated, and exactly one node can make progress at any point. Clearly, the algorithm constructs a DFS tree of the graph connected to the root. A similar strategy of using a vector variable in a message has been employed by Finn [4] in developing resynch procedures.

The algorithm above assumes that the total number of nodes in the network is known to all nodes a priori. When $|V|$ is not known, the algorithm is modified as follows. Whenever a node P_i receives the message for the first time, it extends M to the largest of the received M array size and P_i's neighbor IDs. Clearly, a node extends the received array only if the array size is smaller than the largest ID of its neighboring nodes. This is done following the statement $M.i$:= true, in the above algorithm at node P_i. The root node, at the start of the algorithm, creates an M array of a size that is the largest of the root ID and IDs of its neighboring nodes.

4. Complexity analysis

In the previous section, we presented the DDFS algorithm. In this section we show that both the time and message complexities of the algorithm are $O(|V|)$.

4.1. Theorem. *The proposed algorithm is optimal in communication complexity and uses exactly* $2|V|$ *messages.*

Proof. Every node in the network receives only one message from its father (forward path) and sends one message to its father (return path). No message is sent to an already visited node in the network. Thus, each of the $|V|$ nodes exchanges messages with its father exactly twice. Hence, the total number of messages used in the algorithm is exactly $2|V|$. The message complexity of the algorithm is $O(|V|)$ and is optimal within a constant. □

4.2. Theorem. *The algorithm terminates after* $2|V|$ *units of time if all messages are delivered in one unit of time.*

Proof. The total time is the time required to transmit the messages over the links. From Theorem 4.1, the total time needed to transmit $2|V|$ messages is $2|V|$ units of time if all messages are delivered in at most one unit of time. □

5. Discussion

Reif [7] showed that the DFS problem is inherently sequential, and our algorithm is developed taking this fact in to account. It is interesting to see that the proposed solution is independent of whether communication between nodes in the network is either synchronous or asynchronous. The power of this algorithm in providing efficient solutions to several well-known problems in distributed systems is explored in [8].

Acknowledgment

The authors thank the anonymous referees for their comments, which improved the presentation of the paper. They also thank the communicating editor for his assistance in preparing the revised version of the paper. Thanks to Dr. Abha Moitra for providing useful comments on an earlier draft of the paper.

References

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

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

[3] I. Cidon, Yet another distributed depth-first-search algorithm, *Inform. Process. Lett.* **26** (1987/88) 301–305.

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

[5] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms* (Computer Science Press, Rockville, MD, 1984).

[6] K.B. Lakshmanan, N. Meenakshi and K. Thulasiraman, A time optimal message-efficient distributed algorithm for depth-first-search, *Inform. Process. Lett.* **25** (1987) 103–109.

[7] J.H. Reif, Depth-first search is inherently sequential, *Inform. Process. Lett.* **20** (1985) 229–234.

[8] M.B. Sharma, S.S. Iyengar and R.L. Kashyap, A unified approach for solving a class of problems in distributed systems, submitted for publication, October 1988.