CORRIGENDA

CORRECTIONS TO A DISTRIBUTED DEPTH-FIRST SEARCH ALGORITHM

Information Processing Letters Vol. 32, No. 4 (1 September 1989) pp. 183–186

Devendra KUMAR

Department of Computer Engineering and Science, Case Western Reserve University, Cleveland, OH 44106, USA

Sitharama S. IYENGAR and Mohan B. SHARMA

Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

Communicated by David Gries
Received 26 February 1990

Keywords: Distributed systems, distributed algorithms, communication graph, depth-first search

An earlier paper [1] presents an efficient distributed depth-first search algorithm, with a time complexity $2|V|$ and a message complexity $2|V|$. The algorithm is derived from the traditional sequential depth-first search algorithm. We point out a few errors in the above paper [1] and the corresponding corrections. These errors are mostly minor coding errors; the overall algorithm indicated in the discussion part of the paper remains correct. A formal proof of correctness of the corrected algorithm is in preparation [2].

(1) In Table 1, page 184, in the last row the time complexity should be $2|V|$, rather than $3|V|$.

(2) The code of the algorithm starting at the bottom on page 184 has several minor coding level errors. To understand the errors, note that the semantics of a *Receive(f, M)* command at process i is somewhat ambiguous. If the corresponding message was sent from a process k via a command "*Send(j, M)* to i", then on the reception of the message, what would be the value of variable f? Below we consider two possible interpretations and show that under each interpretation, the code is incorrect.

(a) Suppose the value of f is j. Then in the code of process i, the value of f would be i (since the *Send* commands in the code are of the form *Send(j, M)* to j). Thus the value of the variable f would NOT be the id of i's parent.

(b) Suppose the value of f is k, i.e., the id of the sender of the message (this interpretation seems to be suggested by the point (4) on page 183). Then we have the following problems:

(i) In the **foreach** statement, process i is looking at the neighbors of its parent f (rather than its own neighbors). This is obviously wrong.

(ii) For any node u, the element $M.u$ is being set to true by every child of node u. (This is not a correctness issue but is obviously undesirable—it affects the simple relationship between this algorithm and the well known sequential algorithm where a node is marked only once. Also, it affects efficiency.)

(iii) If a node u has no children, then no process would be setting $M.u$ to true. Thus, depending upon the semantics of the **foreach** statement, the parent of node u may get into an infinite loop, or two different nodes may have the node u as a child (since the element $M.u$ always remains false).

Below we state the corrected code for any process i. We use a *Receive* command of the form "*Receive(M)* from any process k" to mean that a message is received, the value of local variable M

is assigned to the value sent in the corresponding *Send(M)* statement, and the variable k is assigned the id of the sender process of the message.

```
{variable declarations to be given as in [1]}

Receive(M) from any process $f$;
{$f$ is assigned the id of the sender}

$s := \{\ \}$;
$M.i :=$ true;
foreach ($j$: $j$ a neighbor of $i \wedge \neg M.j$:
    $s := s \cup \{j\}$;
    Send(M) to $j$;
    Receive(M) from any process $k$;
    {we are guaranteed that, for the given instance
     of DFS computation, we will have $k = j$})
Send(M) to $f$
```

(3) The paragraph after the above-mentioned code in the paper describes how the algorithm starts. This description requires minor corrections. The root process has to execute a slightly different code (for example, it should not execute the first *Receive* or the last *Send* statement in the code). Also, root's f variable should contain a value such as 0 to indicate that it has no father; note that in the second approach suggested for starting the DFS computation, the root node would have its f variable pointing to another node in the graph, which is obviously not desirable.

To take care of the above issues, we suggest several approaches below.

(a) Suppose the DFS computation is started by a process i on receiving a START signal from the outside world (where i is to become the root of the DFS tree). Then i will not execute the first *Receive* command in the above code; instead it will execute $M :=$ false; $f := 0$ and then the rest of the above code following the first *Receive* statement in the code. Also, instead of executing the last command in the code, i.e., the statement "*Send(M)* to f", it will send a message to the outside world indicating termination of the DFS computation, or start some other computation, etc.

(b) Alternatively, a process i would decide to start the DFS computation starting at some root node, and would send a START message to the root and then execute the above code shown. On receiving the START message, the root process will behave as mentioned in item (a) above.

(c) Alternatively, similar to the second approach discussed in the paper, the above code can be modified so that each *Send* message also carries the id of the father of the receiver node, in addition to M. This value in the message would be assigned to the variable f at the receiver process. Then, the process i that starts the DFS computation would send (0, M) to the root. Further details in this approach are straightforward and are skipped here.

References

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

[2] S.S. Iyengar, D. Kumar and M.B. Sharma, Correctness proof of a distributed depth-first search algorithm, In preparation.