Theoretical Computer Science 120 (1993) 261–278
Elsevier

An optimal distributed algorithm for recognizing mesh-connected networks*

Rajanarayanan Subbiah** and Sitharama S. Iyengar

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

Sridhar Radhakrishnan

School of Electrical Engineering and Computer Science, University of Oklahoma, Norman, OK 73019, USA

R.L. Kashyap

Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907, USA

Communicated by M. Nivat
Received August 1991
Revised April 1992

Abstract

Subbiah, R., S.S. Iyengar, S. Radhakrishnan and R.L. Kashyap, An optimal distributed algorithm for recognizing mesh-connected networks, Theoretical Computer Science 120 (1993) 261–278.

In this paper, we consider the problem of recognizing whether a given network is a rectangular mesh. We present an efficient distributed algorithm with an $O(N)$ message and time complexity, where N is the number of nodes in the network. This is an improvement of a previous algorithm presented in Mohan (1990) with a message complexity of $O(N \log N)$ and time complexity of $O(N^{1.6})$. The proposed algorithm is constructive in nature and also assigns coordinates to the nodes of the network.

Correspondence to: S.S. Iyengar, Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA.

* This research, in part, is partially supported by Board of Regents (LEQSF-RD-A-04, 1990) grant and by ONR N000014-91-J1306.

** Present address: Hewlett-Packard Company, MS 47 UE, 19447 Pruneridge Avenue, Cupertino, CA 95014-9913, USA.

0304-3975/93/$06.00

1. Introduction

The problem of network recognition in an asynchronous network has been explored extensively in the past. In the literature on distributed algorithms, there are many efficient algorithms for the recognition of trees, complete graphs, rings, star and bipartite graphs [15]. The previous result for the recognition of mesh-connected networks is one using $O(N \log N)$ messages and $O(N^{1.6})$ time units [12]. The bottleneck in that algorithm is the use of a breadth-first search tree [3]. The proposed algorithm does not use a BFS tree in the recognition process. Our algorithm also assigns labels (coordinates) to the nodes which enables each node to know its exact position in the grid. Throughout this paper, we shall use the words mesh, rectangular mesh and grid interchangeably.

Many interesting distributed algorithms have been proposed in the recent past relating to graph-theoretic problems, such as shortest paths [4, 6, 16], minimum-weight spanning trees [2, 9, 14], biconnected components [5], centers [10] and knots [11]. Many problems such as message routing, center finding, etc., can be solved very easily by imposing certain restrictions on the structure of the network. In general, algorithms for general networks are expensive in terms of both time and message complexities, whereas algorithms for certain topologies are much faster and easy to design. It is shown in [7] that problems such as shortest path, routing, etc., become easy to handle if the network is planar. Message routing becomes easy to handle on mesh-connected or planar networks. Hence, recognition algorithms can be used as a *preprocessing* step in the design of efficient algorithms for problems on networks. Therefore, an obvious consideration would be that the cost of this preprocessing is reasonably smaller than the cost of the general algorithm. This is the basic motivation of the proposed work.

2. The model

An asynchronous network is a point-to-point (or store and forward) communication network, described by an undirected communication graph $G=(V, E)$, where the set of nodes V represents processors of the network and the set of edges E represents the bidirectional noninterfering communication channels operating between neighboring nodes. There is no shared memory between processors in the system. Each node processes messages received from its neighbors, performs local computation, and sends messages to its neighbors, all in negligible time. The communication complexity C is the worst-case total number of elementary messages sent during the execution of the algorithm. The time complexity T is the maximum possible number of time units from the start to the completion of the algorithm, assuming that the intermessage delay and the propagation delay of an edge is at most one time unit. The model assumed is a common model for communication networks [1, 8]. We will also assume that an arbitrary node in the network initiates the recognition algorithm.

3. What is a mesh?

In the rest of the paper, we consider only the case of nontrivial networks, with N, the number of nodes, being at least 4. The trivial case of a network being a single node or a path of nodes in not considered as it is not interesting.

3.1. *Necessary conditions*

A mesh network (see Fig. 1) must consist of vertices with degrees 2, 3 and 4 only, and with exactly four 2-degree vertices. Let D_2, D_3 and D_4 denote the number of 2-degree, 3-degree and 4-degree nodes, respectively. Let the dimensions of the mesh be (m, n), where N (the total number of nodes in the network) $=(mn)=D_2+D_3+D_4$. The parameters of the mesh, namely, D_2, D_3, D_4, N, m and n, should satisfy the following conditions:

total number of nodes $N=mn$,

$D_3+D_4=N-4$,

number of 3-degree nodes, $D_3=2(m+n-4)$,

$$D_3=2\left(\frac{N}{m}+m-4\right).$$

Solving for m, we get

$$m^2-m\left(\frac{D_3}{2}+4\right)+N=0. \qquad (*)$$

Definition 3.1. A nontrivial network is said to be an (m, n) *mesh-looking* structure (Fig. 1) if (*) yields two positive (not necessarily distinct) integer solutions.

Remark 3.2. An (m, n) *mesh-looking* structure need not be a rectangular mesh.

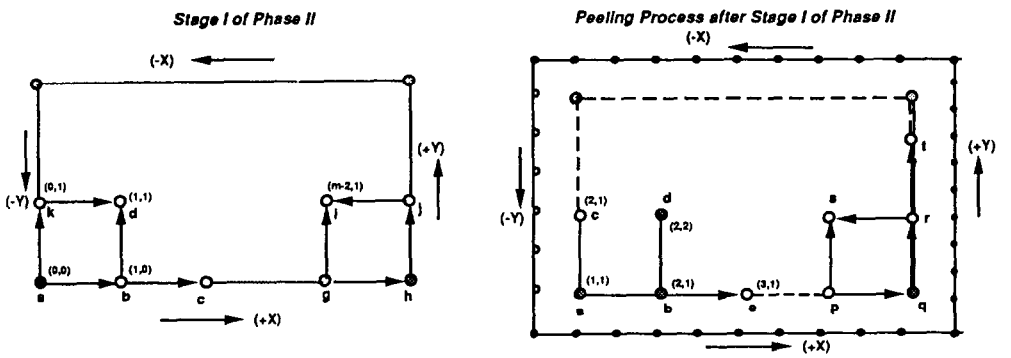

Fig. 1.

Lemma 3.3. *An (m, n) mesh-looking structure with N nodes has $O(N)$ edges.*

Notation. For $n \geqslant 1$, let $[n]$ denote the set $\{0, 1, \ldots, n-1\}$.

3.2. Sufficient conditions

Theorem 3.4. *An (m, n) mesh-looking structure is a rectangular mesh iff there exists an injective map $F: V \to [m] \times [n]$, such that for any $u, v \in V$ with $F(u) = (i, j)$ and $F(v) = (i', j')$, whenever $uv \in E$,*

$$|i - i'| + |j - j'| = 1.$$

It can easily be verified that, except for symmetry, the mapping F is unique. For any 2-degree node v, $F(v) \in \{(0,0), (m-1,0), (0, n-1), (n-1, m-1)\}$. Once this is fixed, because of adjacency constraints, a 3-degree node can assume labels from the set $\{(1, 0), (2, 0), \ldots, (m-2, 0), (0, 1), (0, 2), \ldots, (0, n-2), (1, n-1), (2, n-1), \ldots, (m-2, n-1), (m-1, 1), (m-1, 2), \ldots, (m-1, n-2)\}$.

4. Informal description of the proposed algorithm

The algorithm consists of two phases. In the first phase, the necessary conditions for a network to be a mesh are checked. If these conditions are met, then the second phase strips the mesh layer by layer, each time leaving behind a mesh of smaller dimension, until an intermediate mesh structure of dimension (1, ?), (2, ?), (2, ?), or (?, 2) is identified, where ? denotes an arbitrary integer. During this phase, nodes in the mesh network are assigned coordinates. A successful assignment of coordinates to the nodes, satisfying the adjacency criteria, ensures recognition of a mesh structure. The algorithm rejects any other structure.

4.1. Phase I

During phase I of the recognition algorithm, the following tasks are performed:

(1) Compute the values of D_3, D_2, D_4, m, n, N.
(2) Check the necessary conditions.
(3) Send information about m and n to all nodes.
(4) Instruct nodes to send *id* and degree to all their neighbors.
(5) Select any 2-degree node and initiate phase II.

In this phase, the values of the various mesh parameters are determined. In order to do this, we construct a depth-first search tree (which is also a spanning tree) of the network using $O(N)$ messages in $O(N)$ time [13]. We shall now informally discuss the working of the algorithm. The root of this spanning tree generates a "Count" message, which has the following format:

$$(D_2, D_3, D_4, N, \text{Flag}),$$

where the Flag field is boolean (presence or absence of nodes with degree other than 2, 3 or 4). This message is trickled down to the leaf nodes, which return the message after updating the values of the various fields. An internal node receiving a return message from all its children updates (counts the number of 2-degree, 3-degree and 4-degree nodes in its subtree) the value of the various fields and sends it to its parent, and this process goes on until the root receives messages from all its children. At this stage the root node determines the values of the various parameters and checks whether the necessary conditions have been satisfied.

If any of the necessary conditions are violated then the algorithm is terminated. The root node sends a message down the tree, containing the expected dimensions of the mesh (m, n). This message is propagated by the internal nodes of the tree, until a leaf node is reached. When a node receives this message about the dimensions of the mesh, it sends information about its *id* and degree to every one of its neighbors, in the network, and also propagates the dimensions down to its children. Then, some 2-degree node is identified, and a message is sent to that node to initiate the second phase of the algorithm. Hence, at the conclusion of phase I, all the necessary conditions have been satisfied and the structure has been identified as a *mesh-looking* structure. Additionally, every node knows the *id* and degree of all its neighbors and the dimensions of the mesh (m, n). The newly selected 2-degree node now starts phase II.

4.2. Phase II

The algorithm described in this section is used only for the recognition of those mesh structures whose dimensions are (m, n), where $m, n > 2$. A slight modification of the algorithm can be used to recognize the special case of $(m, 2)$ or $(2, n)$ mesh structures. The algorithm for the recognition of mesh structures with dimensions $(m, 2)$ or $(2, n)$ is presented in Section 8.

Each node has three variables, namely, its coordinate (x, y), static degree D_s and dynamic degree D_d. Initially, D_d is equal to D_s, the number of neighbors of a node. Each node also maintains an active neighbor list N_1 of all its neighbors and their static degrees. The algorithm in this phase is constructive in nature. This phase is divided into many stages. At each stage, a layer of the mesh is peeled (much like an onion peel!). This peeling of a particular layer affects the N_1 list and dynamic degrees of the nodes in the next inner layer. Stage I differs from the other stages in that $D_d = D_s$ for all the nodes participating in this stage. The peeling of a layer is broken into four different directions, namely, $(+x, +y, -x, -y)$. These directions aid in the generation of coordinates for the nodes. The order of the directions is quite important, but it does not matter as to which direction is chosen first. The need for the direction will be clarified in subsequent sections.

4.2.1. Stage I of phase II

At the beginning of phase II, each node sets all the members of its neighborhood list to be ACTIVE. A node enters the DEAD state when its neighborhood list becomes

empty. A successful termination of the algorithm results in every node in the network reaching the DEAD state. A node updates its neighborhood list when a message is either sent or received by that node. The process of updating is the deletion of that member from/to whom the message was received/sent. The process of peeling the outermost layer is initiated by the 2-degree node selected at the end of phase I. This node sends a massage to one of its 3-degree neighbors. This message has the following format:

$$\{\text{Message\_Type}, id, S_{\text{num}}, \text{Coordinates}(x_i, y_i), \text{Dir}\},$$

where S_{num} denotes the stage number ($=1$ in this case), and Dir can be any one of $\{+x, +y, -x, -y,\}$. The actual message sent in this case would be

$$\{\text{"Propagate"}, id, 1, (0,0), +x\}.$$

When a 3-degree node receives this message, it sets its coordinates to $(1,0)$, based on the coordinate of the sender $(0,0)$ and the direction $(+x)$. The rest of the description of stage I will be illustrated with an example as shown in Fig. 2a.

Node b has received the message from node a. Node b has two other neighbors, namely, c and d, which have a degree of 3 and 4, respectively. Node d does not participate in the peeling process, but learns its coordinates based on the message it receives from b. The message sent by node b to node d is

$$(\text{"Propagate"}, \text{b}, 1, (1,0), +y).$$

Node d, on receipt of this message, sets its coordinates to $(1,1)$. In this stage all direction information is taken as is for the computation of the coordinates, whereas in subsequent stages the 4-degree nodes will have to interpret the directions.

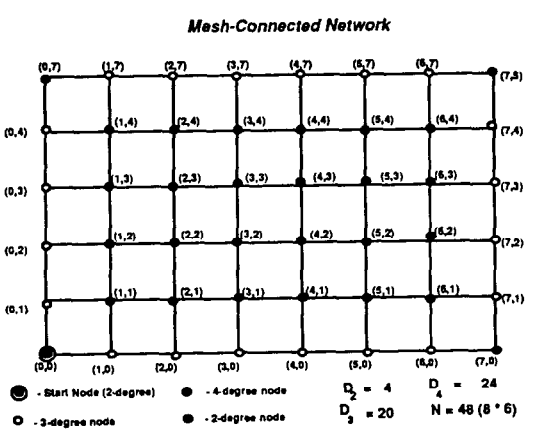

Fig. 2a.

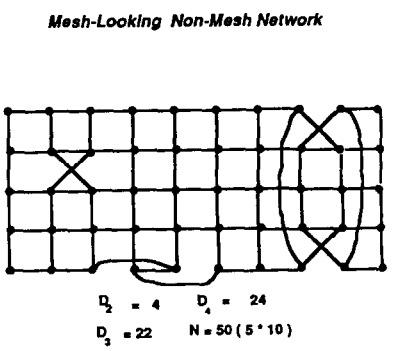

Fig. 2b.

Node b sends the following message to c:

("Propagate", b, 1, (1, 0), $+x$).

Node c, upon receiving this message, sets its coordinates to (2, 0) and propagates this message to its other 3-degree neighbor, who performs similar actions as b.

This process is continued until node g as shown. Node g does not have a 3-degree neighbor other than the one from which it received the propagation message. Hence, it sends the updated propagation message to its 2-degree neighbor. Node g also sends a message to its 4-degree neighbor i, which sets its coordinates to $(m-2, 1)$. If a node receiving a message already has its coordinates set, then the information from the message received is used for confirming its coordinates or for rejection. In case of a rejection, the node sends a special message to its parent in the spanning tree, to terminate the recognition process. In subsequent stages, a node simply sends a "Disagree" message to the originator of the message, who must decide whether or not to terminate the process. It is to be noted that node i receives messages both from nodes g and j. It is immaterial as to which message is received first.

Node h, upon receiving a propagate message from node g, sets its coordinates, and verifies the dimension of the mesh based on its coordinates and the stage number. Node h also changes the direction from "$+x$" to "$+y$", and sends the following message to j:

("Propagate", h, 1, $(m-1, 0)$, $+y$).

This process continues until node k sends a message to nodes d and a. Node d, upon receipt of this second message, confirms its coordinates and initiates the second stage of phase II. At each stage i, the node with coordinates $(i-1, i-1)$ and $|N_1| = 2 (D_d = 2)$ will initiate the peeling process of that stage.

4.2.2. Rest of phase II

The second stage of phase II is initiated by the node which has the coordinates (1, 1) and a dynamic degree of 2. It is important to understand the fact that each node only knows the static degree of its neighbors. The nodes which already have a coordinate assigned in the previous stage participate in the peeling process, and the nodes in the next layer learn and set their coordinates which will be used in the next stage. Hence, the peeling of one layer is not completely independent from the peeling of the previous or next layer. This transfer of information to the next layer ensures the proper working of the algorithm.

When a particular layer of the network is being peeled, the nodes in the next layer (4-degree nodes usually) learn their coordinates. They compute their coordinate value based on the values of the various fields of the received message. Since they are 4-degree nodes, they do not propagate this message. The propagation of the peeling message is done by the dynamic 3-degree and 2-degree nodes. Since the 4-degree nodes do not propagate the message, they know that the direction field of the message

received is wrong, and hence use the *succ*(Dir) to compute their coordinate values. These computed values are then confirmed when the next layer is peeled, and when these nodes become 3-degree nodes they are used to propagate the peeling message.

Every node, upon receiving a message, computes a coordinate value. If its coordinates are not already known, then the newly computed values are its coordinates. If its coordinates are already set then the newly computed value should equal its coordinates. If the node is a 3-degree node then it confirms its coordinates with the newly computed values and continues the propagation process. If it finds that the two values are not the same, then it recomputes the coordinates with *succ*(Dir) and then checks if its coordinates are equal to this newly computed value, as in the case of node s in Fig. 2b. If the values are found to be equal, then it sends a "Disagree" message to the sender, indicating that there is a problem. If the values are found to be different then this node recognizes that the network is not a rectangular mesh and, hence, sends a special "Reject" message to its parent in the spanning tree constructed during phase I. We shall explain the working of the algorithm by identifying three cases, as shown in Fig. 2b.

(1) Node a has b and c as active members of its N_1 list. Each of these nodes has a static degree of 4 and a dynamic degree of 3. Node b has a coordinate of (2, 1) and c has a coordinate of (1, 2). To ensure the same direction of peeling in this layer, as in the previous layer, only b has to propagate the message. a sends the following message to both b and c:

("Propagate", a, 2, (1, 1), $+x$).

Node c will reject the message since the computed coordinate does not match its coordinate. It then sends the following message to a, informing it about the disagreement in the computed coordinate values:

("Disagree", c, 2, (1, 2)).

Node a will ignore this message, since it expects to receive a "Disagree" message from one of its active neighbors. If a node receives "Disagree" messages from all its active neighbors then it sends a special "REJECT" message to its parent in the spanning tree used in phase I.

(2) Node b has two active members in its N_1 list, namely, d and e, both with a static degree of 4. Node e has a dynamic degree of 3, whereas d has a dynamic degree of 4. Node b sends the following message to both d and e:

("Propagate", b, 2, (2, 1), $+x$).

The interpretation of this message by d and e is quite different.

"d": Since d does not have its coordinates set, it learns about the value of its coordinates from the message it received. This computation is not the same as in phase I. The message received by d is a "Propagate" message, but its dynamic degree is not 3. So, it assumes that this message was wrongly sent. To compute its coordinates, d chooses the direction following that which was received in the message, from the set

$\{+x, +y, -x, -y\}$. In this case the direction chosen would be $+y$. Hence, d would set its coordinates to (2, 2), and send a "Disagree" message to b.

"e": Node e already has its coordinates set, and simply confirms that the computed value is the same as its coordinates. Then e follows the same procedure as b.

(3) Node s receives two messages, one each from p and r. The message received from p is $(p, 2, (m-3, 1), +x)$. Since s is a 4-degree node, it computes its coordinate value with the direction changed to $+y$, $(succ(+x))$. The message received from r is (r, 2, $(m-2, 2)$, $+y$). At this point in time, s would have a dynamic degree of 3, since it has already received the message from p. So, it computes its coordinate value with the same direction field value as that received. It tries to confirm its value, but when it finds that the computed value is different from its previously computed coordinates, it recomputes the new value with *succ*(Dir). It is immaterial, for the successful execution of the algorithm, as to which of these two messages reaches s first. Node s learns its coordinates from the first message and confirms this with the second message. If the learned value and the computed value are not the same, then a special "REJECT" message is sent to its parent in the spanning tree. The algorithm is then terminated.

The exact working of the algorithm at each node, upon receiving a message, is provided in Section 5.

During any stage of the peeling process, there exist 4 nodes, whose dynamic degree is 2. One of these initiates the algorithm for that stage/layer. The other 3 perform the following functions:

- The direction field of the next "Propagate" message is changed to that which appears next (successor) in the list $\{+x, +y, -x, -y\}$.
- The current dimensions of the mesh are confirmed based on its coordinate value and the stage number.

This process of peeling is stopped when a degenerate intermediate mesh structure of dimension (1, ?), (?, 1), (2, ?) or (?, 2) (where ? is an arbitrary integer) is identified. This decision is made by the 2-degree node which initiates the peeling process at that stage, since it has knowledge of the current dimensions of the mesh. These special trivial structures will be recognized using a slightly different algorithm. In the following section a pseudo code version of the algorithm is presented. The algorithm is initiated at a particular 2-degree node. Other nodes in the system are initially INACTIVE. When a node receives a message, it executes the following algorithm. In general, messages have the following format:

(Mesg_Type, *id*, *Stage_num*, Coordinates (x, y), *Direction*).

Messages are treated as record structures for the purpose of the description of the algorithm. The field Mesg_Type can take the following values.

"Propagate"

The peeling process in any stage is propagated from one node to another using the "Propagate" message.

"Disagree"

When a node receives a "Propagate" message and finds that the computed value of its coordinates does not agree with its learned value, then it sends a "Disagree" message to the node from which the original message was received.

"REJECT"

When a node is able to determine that the structure is not a rectangular mesh, then it sends a special "REJECT" message to its parent in the spanning tree. This message indicates that the algorithm is to be terminated and, hence, this message is propagated by its parent up to the root.

5. The recognition procedure

The following code is executed by a node i receiving a message *msg* from its neighbor. Each node has the following local variables:

(x_i, y_i)	coordinates of the node,
set	= true if coordinates have been set, else false,
N_{dis}	number of "Disagree" messages received,
N_1	a list of neighbors, initially all ACTIVE,
(M, N)	dimensions of the mesh.

```
Procedure Recognize_Mesh (i, msg);
  i: id;
  msg: message;
/* The following code is executed by a node i receiving a message msg from its
neighbor. Also let P(i) be the parent of node i in the spanning tree constructed in
phase I */
begin
Case msg. type of
"Propagate":
  If stage 1 then
  begin
    If coordinates set then
      If old value ≠ new computed value then
        send ("REJECT", P(i));
      else
        remove sender from active neighbors list
        If (|N1| = 2) and (xi, yi) = (msg.stage, msg.stage) and NOT degenerate then
        /* checking for an intermediate case of (2, *) or (1, *) */
          for every neighbor j do
            Send ("Propagate", i, msg.stage + 1, +x);
            remove recipient from active neighbors list
            {Start Next Stage}
```

```
  else /* coordinates not set */Begin
    Compute new coordinates (same direction)
    Case dynamic degree Of
    3: remove sender from active neighbors list
    for every neighbor j do
    Case static degree of neighbor j Of
      2,3: send ("Propagate", i, 1, (x_i, y_i), msg.Dir);
        remove recipient from active neighbors list
      4: send ("Propagate", i, 1, (x_i, y_i), succ(msg.Dir));
        remove recipient from active neighbors list
    end; {Case Statement - N_1(i)(j).degree}
  4: remove sender from active neighbors list
  2: remove sender from active neighbors list
    To your neighbor do
    Send ("Propagate", i, (x_i, y_i), succ(msg.Dir));
    If Not Dimension_Confirm then
      send ("REJECT", P(i));
    end; {Case Statement - |N_1|}
  end;
end
else {Stage No: ≥2}
  If coordinates not set then
    If |N_1| = 4 then
      (x_i, y_i) = Compute (msg.(x, y), succ(msg.Dir));
      Send ("Disagree", msg.id);
      remove sender from active neighbors list
    else
      Send ("REJECT", P(i));
  else{Coordinate Set During Previous Stage}
  begin
    If (x_i, y_i) = Compute (msg.(x, y), msg.Dir) then
      remove sender from active neighbors list
      for every active neighbor j do
        Send ("Propagate, i, msg.stage, (x_i, y_i), msg.Dir);
        remove recipient from active neighbors list
        {Continue Same Stage}
    else
    If (x_i, y_i) = Compute (msg.(x, y), succ(msg.Dir)) then
    begin
      Send ("Disagree", msg.id);
      {When current dimension becomes (2, *)}
      If |N_1| ≠ (2 or 3) then
      remove sender from active neighbors list
```

```
        If |N1|=2 then
          If (xi, yi)=(msg.stage, msg.stage) and NOT degenerate then
            for every neighbor j do
              Send ("Propagate, i, msg.stage+1, (xi, yi), +x);
              remove recipient from active neighbors list
              {Start Next Stage}
        else If (degenerate) then
          /* An intermediate mesh of dimension
            (1,*), (*,1), (2,*) or (*,2) is identified.*/
          Recog_degenerate; /* in the next section */
      end
      else
        send ("REJECT", P(i));
    end; {End PROPAGATE}
"Disagree":
    Ndis=Ndis+1;
    If Ndis=2 then
      Send ("REJECT", P(i));
"REJECT":
    If not root (Spanning Tree) then
      Send ("REJECT", P(i))
    else
      Start (Terminate (REJECT));
end; {End of Case Statement}
end; {End of procedure Recognize_Mesh}
```

The following procedure is followed when a degenerate case is identified during the recognition procedure. Note that these procedures are used only when an intermediate mesh of dimension (1,*), (*,1), (2,*) or (*,2) is identified.

The computation of the coordinate value is performed locally by each node, upon receipt of a message, by using the procedure "Compute".

```
Procedure Compute ((x, y), Dir);
input: Sender's coordinate value and direction
output: New coordinate values
begin
  Case Dir of
  "+x": return ((x+1, y));
  "+y": return ((x, y+1));
  "-x": return ((x-1, y));
  "-y": return ((x, y-1));
  end;
end;
```

One of the following procedures are executed when the procedure Recog_degenerate is called from the procedure Recognize_Mesh.

```
Procedure Recognize_Intermediate_(1,?)_Mesh (i, msg)
  i: id;
  msg: message;
begin
  If coordinates are not set then
    send ("REJECT", P(i));
  else
  If message type = "X dimension = 1" then
    remove sender from active neighbor list;
    If active neighbor list not empty then
      send to unique active neighbor
      ("X dimension = 1", i, msg.stage, (x_i, y_i), msg.Dir);
    else
      Send ("FINISH", P(i));
end;
/* A similar procedure is followed if the "Y dimension is 1" */

Procedure Recognize_Intermediate_(?,2)_Mesh (i, msg);
begin
  If coordinates not set then
    Send ("REJECT", P(i));
  else
  If message type = "X dimension = 2" then
    If msg.(x, y)∈{ } then
      To both active neighbors do
      send ("X dimension = 2", i, msg.stage, (x_i, y_i), NO Direction)
      remove sender from active neighbor list;
      If (active neighbor list is empty) and (x_i, y_i) = (M − m.stage, m.stage − 1) then
        send ("FINISH", P(i));
end;
/* A similar procedure is followed if the "X dimension is 2" */
/* The reporting node has a coordinate of (M − 1 − m.stage, N − 1 − m.stage) */
```

6. Termination of recognition

The algorithm is terminated when a message of type "FINISH" is received by the root of the spanning tree. If the root node is in the "DEAD" state then it sends a "Check" message to all its children. Any node receiving the "Check" message will either propagate it down to its children, if its state is "DEAD", i.e. its coordinates are

set, or send a "No" message to its parent. When the leaf nodes receive this "Check" message, they send a "Yes" if the state of the node is "DEAD" and a "No" otherwise.

If a node receives a "No" message then it propagates this message to its parent. If all children send a "Yes", then it propagates a "Yes" to its parent. If the root node receives a "Yes" from all its children, then the network is a rectangular mesh.

7. Proof of correctness

In order to show the correctness of the algorithm, we have to show that the necessary and sufficient conditions for a rectangular mesh (specified in Section 3) are checked by the algorithm. Clearly, in phase I all the necessary conditions are checked by the algorithm. In order to show that the algorithm correctly checks the sufficient conditions, we have to show that

(i) the algorithm correctly issues "Reject" messages if the network is not a rectangular mesh, and

(ii) the algorithm correctly labels the nodes of the network as required by Theorem 3.4.

For the case of (i), our algorithm issues a "Reject" message if during its execution we have the following cases:

(1) In stage 1, if the old coordinate value (label) is not equal to the newly computed value.

(2) Based on the stage number, the computed dimension by a node (with dynamic degree two) does not agree with the dimensions of the network calculated in phase I.

(3) In stages greater than one, if nodes with dynamic degrees equal to two or three do not have their coordinates set.

(4) In stages greater than one, if the old coordinate value is not equal to the newly computed value with msg.Dir, or if the old coordinate value is not equal to the newly computed value with $succ(msg.\text{Dir})$.

(5) When a node receives more than one "Disagree" message from its neighbors.

It is easy to verify that in all the above cases the network is not a rectangular mesh. Now, what remains to be shown is that if none of the nodes in the network, during the execution of the algorithm, satisfies the above conditions, then the network is indeed a rectangular mesh. In order to prove this, we have to show that our algorithm

(a) assigns distinct labels (coordinate values) to all the nodes in the network, and

(b) assigns, for any two nodes u and v in the network, coordinate values (i, j) and (i', j'), respectively, such that, whenever there is a link between nodes u and v in the network, we have $|i-i'|+|j-j'|=1$.

Now from Theorem 3.4 we can show that our algorithm correctly recognizes a rectangular mesh. In order to show (a), we have the following. Let us consider the case of a node with coordinates (p, q). No neighbor of this node is assigned the same coordinate value. Without loss of generality, let us assume that the direction of propagation is $+x$. Any node receiving its coordinates in the same direction has the

first coordinate element greater than p. For this coordinate to decrease, two changes in direction should take place. But the first change in direction would change the second coordinate element. With this simple argument, we can establish that no two nodes can have the exact same coordinate values. The condition (b) is trivially enforced, since our algorithm sets the coordinate values sequentially and the values once set are never changed.

8. Informal discussion of (2, ?) mesh recognition

It can be seen that the algorithm described in the previous sections does not recognize the specific case of a (2, ?) or (?, 2) mesh. Here we shall describe a different algorithm by illustrating it with an example as shown in Fig. 3.

The algorithm will recognize a mesh structure, by assigning coordinates to the nodes, just as we did in the previous algorithm. If, during the process of assigning coordinates, a nonmesh structure is recognized, then the algorithm is terminated. The algorithm works by assigning coordinates to two nodes at a time. We shall give an informal description of the working of the algorithm.

It is assumed that all nodes know the *id* and the degree of their neighbors. During phase I, all nodes also know the expected dimensions of the mesh. The only possibility of the existence of a nonmesh structure is shown in Fig. 3. The algorithm basically checks to see that there are no "cross-connections".

To start with, a 2-degree node is identified, which starts the recognition process, by setting its coordinates to (0, 0), and sending a message to its unique 2-degree neighbor. We shall describe the rest of the algorithm by the example shown in Fig. 3.

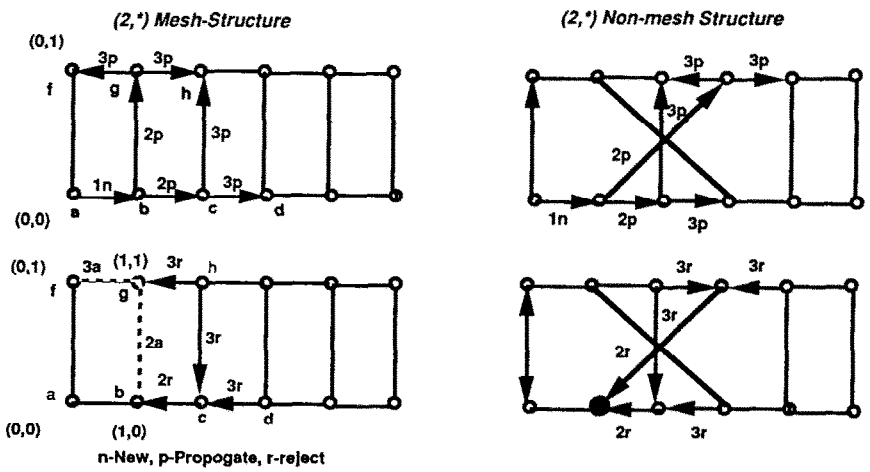

Fig. 3.

Node f, upon receiving a message from node a, sets its coordinates to (0, 1), and sends an "Accept" message back to node a. The next step is to assign coordinates to nodes b and g, and then to nodes c and h, and so on...

Node a sends the message ("New", 1, (0, 0), a) to node b. When node b receives this message, it sends a "Propagate" message ("Propagate", 2, (0, 0), b) to nodes g and c. Node c, upon receiving this message, propagates it to its active neighbors d and h, by sending the message ("Propagate", 3, (0, 0), c) to them. A node receiving a message with a distance field of 3 does the following:

(1) If its dynamic degree is 2 and it has its coordinates set, then it sends an "Accept" message to the sender.

(2) Otherwise, it sends a "Reject" message to the sender.

In this case, node c receives two "Reject" messages, upon which it propagates this "Reject" back to node b. Node g receives a "Reject" from h and an "Accept" from node f. Since node g expects to receive only one "Accept" message, it sends an "Accept" to node b, after removing f from its active neighborhood list, and setting its coordinates to $(0+1, 0+1)$. Node b, upon receiving an "Accept" from g, sets its coordinates to $(0+1, 0)$, and sends the message ("New", 1, (1, 0), b) to node c and the process goes on, until node x receives a message of type "New", where it sets its own coordinates and sends a special message to its unique 2-degree neighbor which sends a "Finish" message to its parent in the spanning tree used in phase I.

In the case of nonmesh structures, both nodes c and g would have received 2 "Reject" messages each and would in turn propagate "Reject" messages to node b. Node b, after receiving two "Reject" messages, would in turn send a "Reject" message to node a, which would terminate the recognition process.

The rest of the algorithm proceeds along the same lines as described above and, hence, needs no more explanation.

9. Complexity analysis

First, we shall establish a lower bound on the number of messages needed to recognize the structure of a network. Any algorithm that recognizes the structure of a network requires at least $\Omega(N)$ messages, where N is the number of nodes in the network. This is true because each node has only local knowledge of its neighbors, and hence at least one message needs to be sent by every node, even to count the number of nodes in the network.

The algorithm presented in this paper to recognize rectangular-mesh-connected networks uses $O(N)$ messages, and has a time complexity of $O(N)$. The algorithm works in two phases. In the first phase, it uses $O(N)$ messages to construct an underlying spanning tree (the depth-first search tree) and then uses at most $2N$ messages to compute the values of D_2, D_3 and D_4 and the dimensions of the mesh. During the construction of the depth-first search tree, the recognition algorithm could send a "Reject" message as soon as a node with degree greater than four is determined.

Table 1

	Previous algorithm	Our algorithm
Time complexity	$O(N^{1.6})$	$O(N)$
Message complexity	$O(N \log N)$	$O(N)$

Using Lemma 3.3, we can see that phase I requires only $O(N)$ messages and can be completed in $O(N)$ time. In the second phase of the algorithm, each node sends at most 3 messages, where the size of the message is fixed and is not dependent on the size of the network. The final phase of terminating the algorithm uses at most $2N$ messages. Therefore, the total message complexity of the algorithm is $O(N)$.

Assuming that every message consumes at most one time unit to reach its destination, it can easily be shown that the algorithm has a time complexity of $O(N)$ (cf. Table 1).

10. Conclusions

We have presented message optimal $O(N)$ distributed algorithms to recognize rectangular-mesh-structured networks. This algorithm is an improvement over a previous one presented in [12], as shown in Table 1. The algorithm not only recognizes mesh-connected networks but also assigns coordinate labels to each node, which can be used for efficient routing. The algorithm described here can be added to the unified optimal distributed algorithms presented in [15] to recognize if a network is a tree, ring, star, complete graph or a bipartite graph. There are other classes of graphs, namely, planar graphs, outer-planar graphs, for which distributed recognition algorithms can be constructed under the same framework.

References

[1] B. Awerbuch, An efficient network sychronization protocol, *J. ACM* **32** (1985) 804–823.
[2] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning trees, counting, leader election and related problems, in: *Proc. Symp. on Theory of Computation* (1987) 230–240.
[3] B. Awerbuch and R.G. Gallager, Distributed BFS algorithm, *IEEE Found. Comput. Theory* (1985) 250–256.
[4] K.M. Chandy and J. Misra, Distributed computation on graphs: shortest path algorithms, *CACM* **25** (1982) 833–837.
[5] E.J.H. Chang, Echo algorithms: depth parallel operations on general graphs, *IEEE Trans. Software Engrg.* **SE-9** (1982) 391–401.
[6] C.C. Chen, A distributed algorithm for shortest paths, *IEEE Trans. Comput.* **C31** (1982) 898–899.
[7] G. Frederickson, A single source shortest path algorithm for a planar distributed network, in: *Proc. Sympos. Theoret. Aspects Comput. Sci.* (1985).
[8] R.G. Gallager, Distributed minimum hop algorithms, Tech. Report, LIDS-P-1175, MIT, 1982.

[9] R. Gallager, P. Humblet and P. Spira, A distributed algorithm for minimum-weight spanning trees, *ACM TOPLAS* **5** (1983) 66–77.
[10] E. Korach, D. Rotem and N. Santoro, Distributed algorithms for finding centers and medians in networks, *ACM TOPLAS* **6** (1984) 380–401.
[11] J. Misra and K.M. Chandy, A distributed graph algorithm: knot detection, *ACM TOPLAS* **4** (1982) 678–686.
[12] S. Mohan, Efficient distributed algorithms for network facility problems, Ph.D. Dissertation, Louisiana State University, Baton Rouge, LA, 1990.
[13] S.B. Mohan, S.S. Iyengar and M.K. Narasimha, An efficient distributed depth-first-search algorithm, *Inform. Process. Lett.* **32** (1989) 183–186.
[14] D.S. Parker and B. Samadi, Adaptive distributed minimal spanning tree algorithms, in: *Proc. 1st Symp. Reliability Distrib. Software Databases* (1981).
[15] K.V.S. Ramarao, Distributed algorithms for network recognition problems, *IEEE Trans. Comput.* **38** (1989).
[16] K.V.S. Ramarao and S. Venkatesan, On finding and updating shortest paths distributively, in: *Proc. 24th Allerton 1986 Conf.*; *J. Algorithms*, to appear.