A DISTRIBUTED SHORTEST - PATH ALGORITHM

Pierre A. Humblet*
Massachusetts Institute of Technology
Electronic Systems Laboratory
Room 35-203
Cambridge, Mass. 02139

ABSTRACT

The problem of routing in a data network is often treated by assigning traffic dependent lengths to the links of the network and routing traffic from node i to node j along the shortest path from i to j . We present a distributed algorithm in which the nodes cooperate to find all shortest paths. It runs asynchronously in every node and does not require the network topology, or even the number of nodes in the network, to be known a priori by the nodes.

INTRODUCTION

The problem of routing in a computer network is often treated by assigning traffic dependent lengths to the links of the network and routing the traffic from node i to node j along the shortest path from i to j. If a central facility (like in TYMNET [1]) monitors the traffic then the shortest paths can be computed at the central location by using classical shortest path algorithms [2]. The difficulty arises when the traffic is measured locally, so that each node knows only the lengths of its outgoing arcs. This is the case of the ARPANET [3] which employs a distributed algorithm to estimate the shortest paths. Errors in such estimates lead to inefficiencies, as explained in [3].

We present an algorithm in which the nodes cooperate to find all shortest paths. It works well when the ratio of the longest to the smallest arc lengths is not too large, and can be seen as a generalization of an algorithm due to Gallager [4] that finds paths containing as few arcs as possible. Other distributed shortest path algorithms have been proposed recently [5], [6], [7]. Comparisons between them awaits simulating them all on comparable networks as worst case behavior is not a reliable indicator of the goodness of an algorithm.

* This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Grant N00014-75-C-1183 and in part by Codex Corporation, Newton, Mass.

We first describe precisely our model and assumptions. This is followed by an explanation of the theoretical basis on which the algorithm rests, including sufficient conditions for its correctness. We then describe the algorithm, explain how it can be optimized and give simulation results.

Description of the Network

The network consists of a finite set N of N nodes, and a set A, included in N x N, of directed arcs. To each arc (i,j) in A is assigned a number (real or ∞) L((i,j)), called the length of arc (i,j). We say that node i is upstream (downstream) of node j if (i,j)$\in A$ ((j,i)$\in A$). A node can be both upstream and downstream of another node.

A chain is a finite sequence of nodes such that each node except the last is upstream of the next node in the sequence. The length of a chain (i_1, i_2 , ... i_n) is defined as $\sum_{j=1}^{n-1} L((i_j, i_{j+1}))$. A loop is a chain that starts and ends with the same node. A path is a chain that contains no loop.

We constraint the L(.)'s to be non-negative, and such that all loops have positive lengths.

D.B. Johnson [8] has noted that algorithms similar to the one given below work even if negative lengths are allowed, but can then have very long running times.

Description of the Initial and Terminal States of Knowledge.

The desired terminal state of knowledge is for each node to know the first arc and the length of a shortest path to each other node at finite distance.

The amount of initial knowledge that the algorithm given below requires to achieve this goal is very small. If initially the computers at all nodes are eventually given a signal to start, the identity of their nodes, the number of downstream nodes and the lengths of the arcs to those nodes, then our objective will be achieved, but the algorithm will not terminate!

In order for the algorithm to stop we require that either one of the following also be known to each node i:

(a) the number of nodes located at finite distance from i

(b) an upperbound LMAX on the length of any arc of finite length.

From a data network point of view it is reasonable to assume that LMAX is given, as the length of an arc is usually represented by a short binary number. In contrast, the number of nodes located at finite distance is usually random, as nodes and links can fail. In the version of the algorithm given below, we assume that LMAX is known.

THEORETICAL BASIS FOR THE ALGORITHM

It is easy to find the lengths and the second and terminal nodes of k + 1 shortest chains starting at a node when knowing the lengths of its outgoing arcs and the lengths and terminal nodes of k shortest chains starting at each of its downstream neighbors: if the set of nodes downstream from node i is N_i, a k + 1st shortest chain starting at i has the form (i,j), jϵN_i, or (i,C_j), where C_j is one of k shortest chains outgoing from a node jϵN_i.

Thus, if the number of chains of length less than x is finite for all x (a sufficient condition for this is that all loops have positive length), then, by generating recursively all chains in order of increasing length, we can find the shortest paths to all nodes located at finite distance, and the first arcs of those paths. The algorithm can stop at a node when the length of the longest known chain is greater than the length of the longest known path + LMAX, as the shortest paths to all nodes located at finite distance are then found.

One might wonder why we do not generate immediately all shortest paths. Unfortunately it is not always possible to find k + 1 shortest paths starting at a node when knowing only the lengths of its outgoing arcs and k shortest paths starting at each of its downstream neighbors. However, it is possible to generate recursively a relatively small set of chains containing all paths of interest, as follows.

At step 0, node i knows the distance to itself (0) and the length $\ell_i^0 := \min_{j \epsilon N_i} (L((i,j)))$ of its shortest outgoing path. It transmits these facts to its upstream neighors. The algorithm proceeds recursively:if at step k + 1 node i has received from all nodes jϵN_i the lengths and destinations of all shortest paths shorter than ℓ_j^k , and node i has also received the ℓ_j^k's, then it can compute $\ell_i^{k+1} := \min_{j \epsilon N_i} (L((i,j)) + \ell_j^k)$. Moreover, let N_{in} be the set of downstream neighbors of i that have transmitted the lengths d_{jn} of their shortest paths to node n. Node i can compute $d_{in} := \min_{j \epsilon N_{in}} (L((i,j)) + d_{jn})$. If $d_{in} \leq \ell_i^{k+1}$ then d_{in} is the length of the shortest path from i to n. Node i finds in d_{in} for all nϵN with non-empty N_{in} and transmits to its upstream neighors the lengths and destinations of all shortest paths discovered during step k + 1, and also ℓ_j^{k+1} .

It is easy to verify that if the length of all loops are positive, then as k grows, ℓ_j^k becomes greater than the length of any finite shortest path. The algorithm can stop when ℓ_j^k has grown by more than LMAX, without any new shortest path having been found.

The algorithm that follows implements what has just been outlined with one important difference: it runs asynchronously in every node. A step at a node i is then the amount of time between two successive transmissions of the ℓ_i's.

THE ALGORITHM

We first describe the computing resources and data structures at each node, and the meaning of the symbols in relation with what was explained previously. We then define the instruction BROADCAST that we will use later, give the initialization and main routines of the algorithm, and show how it can be improved.

Description of the Computing Resources

Each node of the network contains a computer capable of adding, subtracting, storing and retrieving numbers, and branching on positive and zero results. We will first assume that the amount of available memory is infinite, but we will show later that at most $N_i(2N + 1)$ plus a few numbers need to be stored in node i, where N_i is the number of nodes that are downstream of node i.

Computers at different nodes need not be synchronized, but we require that computers be able to write into the memory of computers located upstream. In the context of data networks, this would be done by having a node send a message to an upstream neighbor; this is easiest when all links are duplex.

Data Structure at Node i

Every node i must have memory space for the following

a) the variables LMAX, N_i and LP. LMAX is defined as an upperbound on the length of an arc of finite length and N_i is tne number of nodes downstream of node i. LP represents the length of the longest known shortest path.
b) the numbers D(j) and the arc index BA(j) , $j \in N$. When the algorithm terminates, D(j) is set to the distance from i to j and BA(j) is set to the index of the first arc on a shortest path to j, if $D(j) < \infty$.
c) the numbers I((i,j)) and 0((i,j)) and the arrays Q((i,j),.) , $(i,j) \in A$. I((i,j)) and 0((i,j)) are "write" and "read" pointers pointing to elements of Q((i,j),). Q((i,j),.) contains the

sequence of chain lengths and chain terminal nodes broadcast by node j, except that Q((i,j),1) is initially set to zero.

The Instruction BROADCAST (B) at Node i

B is either a distance or a node label. In every node j such that (i,i)ϵA:

B1 Q((j,i), I((j,i)) + 1)← B

B2 I((j,i)) ← I((j,i)) + 1

It is important that instruction B2 be executed after instruction B1 as can be seen by examining lines M9 to M11 of the main routine below.

The Initialization Routine at Node i

```
LP ←0
Q((i,j),1)    ← 0    (i,j)ϵA
I((i,j))      ← 1    (i,j)ϵA
O((i,j))      ← 1    (i,j)ϵA
D(j)          ← ∞    jϵN
D(i)          ← 0
BROADCAST  (i)
IF (N_i = 0)  then
              begin
              BROADCAST (∞)
              stop
              end
go to main routine
```

The Main Routine at Node i

```
M1  Find ĵ∈N_i such that  L((i,ĵ)) + Q((i,ĵ), 0((i,ĵ)))  =  Find next shortest chain in set
                                                           min  L((i,j)) + Q((i,j), 0((i,j)))
                                                           j∈N_i
M2  x ← L((i,ĵ)) + Q((i,ĵ), 0((i,ĵ)))                      x is its length
M3  if (x > LP + LMAX) then                                Check if can stop
M4          begin
M5          BROADCAST (∞)
M6          stop
M7          end
M8  if (0(i,ĵ)) = I((i,ĵ))) then BROADCAST (x)             End of step, x = ℓ_i
M9  while (0((i,ĵ)) = I((i,ĵ))) wait                       Wait for transmission from neighbor
M10 0((i,ĵ)) ← 0((i,ĵ)) + 1
M11 y ← Q((i,ĵ), 0((i,ĵ))                                  Read new data
M12 if (y is a  length)  then go to M1                     length or destination?
M13            else
M14            begin
M15            if (D(y) > x) then
M16                          begin                         New shortest path
M17                          BROADCAST (x)                 Broadcast its length x
M18                          BROADCAST (y)                 Broadcast its destination y
M19                          D(y) ← x                      Record the length
M20                          BA(y) ← (i,ĵ)                 Record the first arc on path
M21                          LP ← x                        Update LP
M22                          end
M23            go to M8
M24            end
```

Minimization of the Communication and Storage Costs

In place of transmitting the lengths x in lines M8 and M17, it is enough to transmit the difference between x and the sum of the differences previously transmitted. As such a difference is not greater than LMAX, it can be represented by a short binary number. Also, differences equal to zero need not be transmitted at all.

The amount of required memory space can be reduced by noting that if two adjacent elements of Q((i,j),·) are lengths, the smallest one can be discarded. Thus Q((i,j),·) need to have size 2N only, as it will contain at most N destinations and N lengths. Moreover, the 0((i,j))-1 first elements of Q((i,j),·) can be discarded, so that typically Q((i,j),·) contains much less than 2N elements and dynamic storage schemes could be used.

SIMULATION RESULTS

Three quantities are important in distributed algorithms: the amount of computation at each node, the amount of communication (number of bits transmitted) on each link, and the time to completion. This last quantity is often dominated by the time it takes to exchange messages between nodes. Thus an algorithm in which many short messages are exchanged

will generally take more time than an algorithm in which few long messages are exchanged, even if their communication costs are equal.

If the smallest arc length is 1, and the largest is LMAX, it is easy to see that the amount of computation at node i is no more than of the order of N_i . N.LMAX. and the amount of communication per link is no more than of the order of N log(N) + N.LMAX. log(LMAX) bits. However, as with other algorithms of this type [10], the typical behavior is much less.

In order to get rough estimates of performances, we have simulated the algorithm under the following conditions. We used the topology of the ARPANET at a time when it had 55 nodes and 69 duplex links [9, Fig. 1]. We assigned to each arc independently a random integer length uniformly distributed between 1 and LMAX. We optimized the output sequence as explained earlier, and divided it into packets, including in a packet the output produced between two "waits" (line M9). The time of transmission of a packet was chosen as deterministic (1 time unit) in one case, and randomly chosen from an exponential distribution of mean 1, truncated at 10, in another case. The algorithm was initiated at a randomly selected node, then each node signaled to its neighbors that it was time to start. Results are summarized below as a function of LMAX. Although their sensitivity to the various assumptions is unknown, the number of destinations and lengths transmitted is encouragingly small, considering that the exact shortest paths are obtained. More precise results await the simulation of a complete network (including the data traffic, and having the arc lengths depend on the measured traffic) or the implementation of the algorithm in a working network.

LMAX	Average Number of packets transmitted per link	Average Number of destinations transmitted per link	Average Number of lengths transmitted per link	Time to Completion Deterministic Transmission Times	Random Transmission Times
1	11.8	55	10.6	20	35.9
10	19.5	55	39.1	36.5	63.4
100	29.7	55	75.4	64.5	76.0

Communications costs are not significantly different for deterministic and random transmission times, we give only their averages.

REFERENCES

[1] Schwartz, Mischa, et al, "Terminal-Oriented Computer-Communication Networks," Proceedings of the IEEE, Vol. 60, November 1972, pp. 1408-1423.

[2] Lawler, Eugene, Combinatorial Optimization; Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[3] Kleinrock, Leonard and H. Opderbeck, "Throughput in the ARPANET-Protocols and Measurement," IEEE Trans. Commun., Vol.COM-25, January 1977, pp 95-104.

[4] Gallager, Robert, Personal Communication, probably also in the folkart.

[5] Segall, Adrian, et al., "A Recoverable Protocol for Loop-free Distributed Routing," Proceedings of the International Conference on Communications, Toronto, Canada, 1978.

[6] Friedman, Daniel, "Communication Complexity of Distributed Shortest Path Algorithms M.S. Thesis in preparation, Massachusetts Institute of Technology, Cambridge, Mass.

[7] Gallager, Robert, Personal Communication.

[8]- Johnson, Donald, "A Note on Dijkstra's Shortest Path Algorithm", J. ACM, Vol. 20, 1973, pp. 385-388.

[9] Gerla, Mario and L. Kleinrock, "On the Topological Design of Distributed Computer-Networks", IEEE Trans. Commun., Vol. COM-25, January 1977, pp. 48,60.

[10] Gilsinn, J. and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees", NBS Technical Note 772, U.S. Department of Commerce, 1973.