DISTRIBUTED BFS ALGORITHMS

(Extended Abstract)

Baruch Awerbuch

Robert G. Gallager

MIT

ABSTRACT

This paper develops a new distributed BFS algorithm for an asynchronous communication network.

This paper presents two new BFS algorithms with improved communication complexity. The first algorithm has complexity $O((E+V^{1.5})\cdot\log V)$ in communication and $O(V^{1.5}\cdot\log V)$ in time. The second algorithm uses the technique of the first recursively and achieves $O(E\cdot 2^{\sqrt{\log V\log\log V}})$ in communication and $O(V\cdot 2^{\sqrt{\log V\log\log V}})$ in time.

1. INTRODUCTION

This paper presents new efficient distributed Breadth-First-Search (BFS) algorithms in an asynchronous communication network. The major application of BFS in networks is construction of short paths (in terms of number of edges) from some destination node to all other nodes in the network. The resulting BFS tree is used then for routing of data from other nodes to that destination. The BFS algorithm may be repeated many times in case that the network's topology changes.

From the point of view of network's performance, it is desirable that the messages of the control algorithms like BFS don't occupy much of the network bandwidth. Thus, we will be interested to construct BFS trees efficiently, in the sense of minimizing the message exchange. Improving efficiency of distributed BFS yields improvement in more complex distributed algorithms, like 0-1 network flow, network connectivity, planar separator, etc., in which BFS appears to be the bottleneck. In short, the BFS appears to be a basic problem in the field of complexity of distributed algorithms.

For a synchronous network model, there exists a fairly trivial BFS algorithm which achieves the lower bounds on the communication and time complexities, namely $\Omega(E)$ and $\Omega(V)$, respectively, where E is the number of edges and V is the number of nodes. In this paper, we confine ourselves to the asynchronous network, which is, unfortunately, a much more realistic model; most existing networks are asynchronous (e.g. ARPANET). The asynchronous nature of our network seems to make the problem of performing BFS surprisingly difficult.

The best previous BFS algorithm for dense networks is given in [A-84]. This algorithm requires $O(V\cdot\log_k V)$ time and $O(k\cdot V^2)$ communication, for any constant $1 < k < V$. The best algorithm in terms of communication for sparse networks is due to Fredrickson [F-85] and has complexity $O(V\sqrt{E})$ both in communication and time. With $E=O(V)$ say, this complexity is $V^{1.5}$ which is substantially better in

0272-5428/85/0000/0250$01.00 © 1985 IEEE

communication and poorer in time than [A-84].

This paper presents two new BFS algorithms. The first new algorithm here has complexity $O((E+V^{1.5})\cdot\log V)$ in communication and $O(V^{1.5}\cdot\log V)$ in time. For dense networks, with $E \geq O(V^{1.5})$, this is within a a factor of $\log V$ of the lower bound on communication complexity, and for all but very sparse networks, this improves on [F-85]. The second algorithm is a recursive extension of the first algorithm. Its complexity is $O(E\cdot 2^{\sqrt{\log V \log\log V}})$ in communication and $O(V\cdot 2^{\sqrt{\log V \log\log V}})$ in time, This algorithm meets the lower bounds both on communication and time except for $2^{\sqrt{\log V \log\log V}})$ factor, which is asymtotically less than any polynomial in V.

2. THE MODEL

An asynchronous network is a point-to-point (or store-and-forward) communication network, described by an undirected *communication graph* (V,E) where the set of nodes V represents processors of the network and the set of edges E represents bidirectional non-interfering communication channels operating between neighboring nodes. No common memory is shared by the node's processors. Each node processes messages received from its neighbors, performs local computations, and sends messages to its neighbors, all in negligible time. The sequence of messages sent on any given edge in given direction is correctly received in *FIFO* (*First –In –First –Out*) order, with finite but variable and unpredictable delay. This is a common model for communication networks [A-84],[G-82],[F-85].

The following complexity measures are used to evaluate performance for distributed algorithms. *The Communication Complexity, C,* is the worst case total number of elementary messages sent during the algorithm, where an elementary message contains at most $O(\log V)$ bits. *The Time Complexity, T,* is the maximum possible number of time units from start to the completion of the algorithm, assuming that the inter-message delay and the propagation delay of an edge is at most one time unit. This is under the provision that the algorithm works correctly *without* this assumption.

3. THE PROBLEM

Given an undirected graph (V,E) and a a *start* node $s \in V$, the *Breadth-First Search (BFS)* problem is to find, for each node $i \neq s$, the length l_i of a shortest path in (V,E) (in terms of the number of edges) from s to i and the immediate predecessor (*parent*) of i on that path. The length l_i is called the "layer number" of i. The edge leading from a node to its parent (child) is called *inedge* (*outedge*). This problem is also known as the *Minimum-Hop Problem* [G-82], [S-83].

In a distributed algorithm for the BFS problem, each node has a copy of a *node algorithm* determining the response of the node to messages received at that node. Namely, the algorithm specifies which computations should be performed and which messages should be sent. Initially, each node is ignorant of the global network topology except for its own edges. The algorithm is initiated by the start node s. Upon completion of the algorithm, the start node enters a given final state. At that time, all other nodes in the same connected component of the network know their (correct) layer numbers, their inedges and their outedges.

4. BASIC KNOWN-CONNECTIVITY ALGORITHMS

Let us first outline a simple BFS algorithm, which will be referred to as the *Basic Coordinated Algorithm*. It operates in successive iterations, each processing another BFS layer. The start node controls these iterations by means of a synchronization process, performed over the part of the BFS tree built in previous iterations. At the beginning of a given iteration i, the BFS tree has been constructed for all nodes in layers $j < i$. The start node broadcasts a message over that tree, which is forwarded out to nodes at layer $i-1$. These nodes send "exploration" messages to all neighbors, trying to discover nodes at layer i. When a node first receives such an exploration message, it chooses the sender as its parent and deduces that it belongs to layer i. It sends back an acknowledgement (ack) with an indication whether the sender was chosen as parent or not. Each node at layer $i-1$ waits until all exploration messages have been acknowledged and then sends an ack to its parent in the BFS tree, indicating whether any new descendents have been discovered. When an internal node gets such acks from all children, it sends an ack to its parent. Eventually, all the acks are collected by the start node and thus layer i has been processed completely. If any nodes have been discovered at that layer, the next iteration $i+1$ is started. Otherwise, the algorithm terminates. This algorithm is quite inefficient in the number of messages and time used for synchronization purposes. If one considers a network with all nodes on a single path of length $V-1$, one sees that the communication and time complexity are each $O(V^2)$. Obviously, the performance of the algorithm degrades as the diameter of the network increases.

One of the major approaches in the past [G-82], [F-85] has been to divide the network into groups of successive layers, called *strips*, and to process these strips one-by-one, synchronizing only once per strip. Our first algorithm, described in the next section, uses a similar approach, but processes strips in a novel way. Before processing a new strip, assume that the BFS tree was constructed for all the previous strips. Each node in the last layer of this already constructed tree will be called a *start* node a the new strip. The problem of processing a strip is in a sense a generalization of the original BFS problem instead of constructing a BFS tree from a *single* start node, we want to construct a *forest* of shortest paths from a set of start nodes.

We now want to distinguish between two *types* of BFS problems on a strip. The first problem, referred to as *known-connectivity* BFS, is to construct a BFS forest on a strip *with* starting knowledge of a spanning forest of the strip, each tree spanning another connected component of the strip. The second problem, referred to as *unknown-connectivity* BFS, is to construct a BFS forest on a strip *without* such knowledge.

We present now a simple algorithm solving the first problem, referred to as the *basic coordinated multiple-source algorithm*, which is a natural extension of the basic single-source algorithm above. The idea is that all the start nodes grow their BFS trees by applying the basic algorithm above. All the start nodes in the same connected component will synchronize through the spanning tree of the component so that they proceed at the same speed, i.e.

perform the same iteration at a given time. In particular, the forest is constructed layer by layer. At the beginning of processing a new layer, say layer i, some leader among the start nodes in a given connected component propagates a message to the other start nodes via the spanning tree and such start nodes propagate the message through the already constructed layers of the forest. The nodes at layer $i-1$ then send exploration messages as before, and acks are collected back to the start nodes and then through the spanning tree to the leader. This process is performed independently in each connected component. Note that the absence of coordination between components causes no harm.

With the above strategy, at most one exploration message is sent in each direction on each edge of the strip. Thus, the total number of exploration messages is linear in number of edges. Also, each node in the strip (counting the strip's start nodes) receives at most one coordination message over the spanning tree from the leader and at most one message from its start node through the BFS forest during one iteration of the algorithm. Finally, there is one ack message for each broadcast and exploration message. Altogether there are 4 messages per node per iteration and 4 messages per edge of the strip.

However, what we really need to solve is the second problem. The coordinated algorithm above cannot be applied to solve the second problem, because spanning forest inside the strip is not available. Conceptually, the second problem appears to be more complex than the first problem because, since it is not clear a priori which nodes are in the new strip, there is no obvious method for coordination between multiple start nodes. It turns out that there is no apparent way to reduce the second problem to the first one in an efficient way. There is an efficient distributed algorithm [GHS-83] available for finding a spanning tree and connected components (with almost linear complexities), but it cannot be used because the nodes in the strip are unknown. One may suggest the following naive solution to the problem: construct a global spanning tree in the network and coordinate between start nodes of the strip through that tree. However, coordination over such "big" tree is too expensive and in fact we end up with an algorithm whose complexity may reach $O(V^2)$.

However, let us assume that, magically, spanning forests are available in each strip, and that we can should only solve the known-connectivity BFS inside each strip. Then the BFS tree for the whole network can be generated by using the above coordinated multiple sources algorithm for each strip and by using the synchronization technique of the basic coordinated algorithm between strips. Each node then receives at most 2 synchronization messages per node for each strip being processed. Suppose that strips are chosen to contain $\sqrt{V}$ layers. Since there are at most $\sqrt{V}$ strips and $\sqrt{V}$ layers inside each strip, altogether we have $O(V^{1.5})$ synchronization messages and in total, $O(E+V^{1.5})$ messages have been sent; this is a considerable improvement comparing to the basic algorithm above. This observation shows how nice it would be to reduce the unknown-connectivity problem to the known-connectivity one. In the next section, we show how this reduction can be performed.

5. UNKNOWN-CONNECTIVITY ALGORITHM

The algorithm presented here is referred to as

bootstrap algorithm. The idea is to perform BFS and the spanning tree algorithms interchangingly, each algorithm supporting the other. In general, the algorithm maintains forest of trees, each of the start nodes belonging to some of these trees. Initially, each tree consists of a single start node. The set of start nodes in the same tree is called a *cluster*; one of these nodes is chosen as the *leader* of the cluster, which coordinates operation of the whole cluster. The idea of the algorithm is to try to achieve greater degree of coordination by merging together as many trees as possible.

The algorithm proceeds in iterations, each consisting of two stages. The purpose of the first stage is to grow independently BFS trees form each cluster while the second stage stitches together neighboring trees. Upon completion of a stage, all the start nodes of a certain cluster coordinate with all the other clusters back through the original start node. Thus, at a certain time, all the clusters perform the same stage of the same iteration.

The first stage is a slight generalization of the above coordinated multiple-source algorithm, which will be referred to as *generalized coordinated algorithm*. It uses the above basic coordinated multiple-source algorithm to build an independent BFS forest from each cluster for a given number of layers. (This process is coordinated by the leader of the cluster via the tree.) Since there is no coordination between different clusters and the network is asynchronous, some clusters might grow their forests much quicker than others. As a result, some node might be improperly seized by a forest A which progresses quickly, while actually it might be closer to a start node in another cluster and thus should belong to a forest B grown by that cluster. This situation is discovered later when B eventually grows up and an exploration message sent by a node in B reaches one of the improperly seized nodes. In this situation, the edge carrying that message is remembered by the adjacent nodes as an *inter-forest* edge and both forests A and B terminate their part in the first stage, even though the required number of layers has not yet been explored.

Upon the completion of the first stage, the cluster and its BFS forest are called *active* if the BFS forest is adjacent to any inter-forest edges; otherwise they are *inactive*. Note that an inactive cluster's BFS forest must contain all nodes whose true BFS paths must go through the start nodes of the cluster, since none of these nodes could be improperly seized by other forests. Thus, if upon completion of the first stage all the clusters are inactive, then the strip has been processed correctly and the bootstrap algorithm terminates.

Otherwise, the second stage is started, in which the spanning tree algorithm [GHS-83] is applied to a super-graph whose nodes are active forests and whose edges are the inter-forest edges. (A more detailed description is given in the full paper.) The output of this algorithm is a spanning forest of the super-graph, each tree spanning another component; in addition a leader is chosen inside each tree. As a result, active forests which share common inter-forest edges are merged together into bigger forests.

In the successive iterations, the first stage is modified as follows. Whenever an active forest meets another *inactive* forest on its way, the former forest penetrates into the latter, absorbing all of its nodes

which were improperly seized. (This could not occur in the first iteration since all the forests are initially active.) If an active forest is met, then, as before, the exploration process is stopped and an inter-forest edge is remembered. The idea behind this rather peculiar rule is to reduce the number of iterations without increasing too much the number of penetrations.

We now observe that after each iteration, each active cluster either becomes inactive or merges with at least one other active cluster. Thus, after i iterations, each cluster not yet inactive contains at least 2^i start nodes. Thus after at most $\log V$ iterations, each cluster is inactive and the resulting forest is a genuine BFS forest for the strip.

In summary, the bootstrap algorithm effectively solves the problem of processing a strip with unknown-connectivity by at most $\log V$ iterations, each involving one application of the generalized coordinated algorithm, two global synchronization procedure and one application of the spanning tree algorithm. Each application of the spanning tree algorithm of [GHS-83] requires $O(\log V)$ messages per node plus $O(1)$ message per edge of the strip. In terms of time, it requires $O(\log V)$ time per node of the strip. (Here we use the fact that the subnetwork, which is the input to the spanning tree algorithm, belongs entirely to the strip.)

Let us now plug the the bootstrap algorithm into the scheme at the end of the previous section, which divided the network into strips of size $\sqrt{V}$ and processed them serially. It is easy to show that the resulting algorithm has communication complexity $O((E+V^{1.5})\cdot\log V)$. Since the exploration messages, which account for the term E above, are sent in parallel at each node, (and similarly for the spanning tree algorithm), the time complexity is $O(V^{1.5}\cdot\log V)$. Note that this algorithm requires knowledge of V; however, both V and E can be easily calculated with complexity $O(E)$ in communication and $O(V)$ in time.

6. $O(E\cdot 2^{\sqrt{\log V \log\log V}})$ ALGORITHM

We now modify the generalized coordinated algorithm above, making it recursive. This modification will be referred to as *main algorithm*. Recall that the input to this algorithm is a set of start nodes, or a cluster, and a spanning tree T of the cluster which belongs entirely to the current strip. In this algorithm, the strip is split into a set of substrips, each with a common number of layers. The algorithm proceeds in successive iterations, each processing another substrip by calling the bootstrap algorithm as a subroutine. The input to this subroutine is a set of start nodes for the substrip plus a structure for external coordination, which is needed in order to trigger the execution of the subroutine, to detect its termination and to synchronize between internal iterations of the subroutine. This structure consists of the BFS forest of the preceding substrips plus the above tree T. The start nodes of the present strip are the nodes on the final layer of the BFS forest. (For the first substrip, this set of start nodes is simply the cluster being used by the main algorithm.)

The subroutine itself proceeds in at most $\log V$ internal iterations, each involving, among the rest, solving known-connectivity problem for *substrips*. This problem will be solved by the *same main algorithm* (instead of the generalized coordinated algorithm, as before). Note that the main algorithm for

strips calls the algorithm for smaller *substrips*, i.e. is performed recursively, decreasing the depth of the strip as the recursion depth increases. At the bottom level of recursion, the basic coordinated multiple-source algorithm is used.

Let us now evaluate the complexity of the main algorithm. Consider the processing of a strip of size d with a known connectivity by the main algorithm above. For convenience, we will consider here the *normalized* complexities of the algorithm, denoted by $CE(d)$ and $TN(d)$, which are upper bounds on the number of messages sent per link of the strip and the time spent per node of the strip, respectively. Observe that the overall complexities of BFS algorithm are $C=E\cdot CE(V)$ and $T=V\cdot TN(V)$. (We assume the worst case, i.e. that the diameter of the network is V.) We will provide here a recursive equation or evaluating for $CE(h)$; it turns out that $TN(h)$ satisfies the same equation and thus $CE(h)=TN(h)$.

Let us denote by d_i the depth of the strip being processed at the i's level of recursion, $V=d_1 > d_2 .. > d_r$, where r is the maximum depth of the recursion. The complexity at the bottom level is $CE(d_r)=TN(d_r)=O(d_r)$. The algorithm processing strips of depth d_i consists of $\frac{d_i}{d_{i+1}}$ iterations, each processing substrips of size d_i, and involving logarithmic number of internal iterations. Each internal iteration involves one call to the algorithm processing strips of size d_{i+1}, one call to a synchronization procedure, and one call to the spanning tree algorithm. Taking these factors into account yields the following recursive formula: $CE(d_1) = \log V\cdot[\frac{d_i}{d_{i+1}} + CE(d_{i+1})]$. It turns out that $TN(d)$ obeys the same recursive formula, and thus $CE(d)=TN(d)$.

Using dynamic programming, one can optimize total the complexities of the resulting algorithm, choosing properly the depth r of the recursion and the numbers d_i. Let us just mention that the optimum solution for very sparse and very large networks has the form $CE(V) = O(2^{\sqrt{\log V \log\log V}})$. Thus, the total communication and time complexities of the algorithm are bounded by $O(E\cdot 2^{\sqrt{\log V \log\log V}})$ and $O(V\cdot 2^{\sqrt{\log V \log\log V}})$, respectively.

REFERENCES

[AG-84] B.Awerbuch and R.Gallager, "An $O(V^{1.6}\cdot\log V+E)$ distributed BFS", preprint, November 1984.

[A-84] B. Awerbuch, "An Efficient Network Synchronization Protocol ", ACM Symposium on Theory of Computing, April 1984, Washington.

[G-85] G. Frederickson, "A single source shortest path algorithm for a planar distributed network", Proc. 2nd Symp. on Theoretical Aspects of Computer Science (Jan. 1985).

[G-82] R.G. Gallager, "Distributed Minimum Hop Algorithms", M.I.T. Technical Report, LIDS-P-1175, January 1982.

[GHS-83] R.G. Gallager, P.A. Humblet and P.M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees", ACM Trans. on Program. Lang. & Systems, Vol. 5, pp. 66-77, January 1983.