1 Overview

We will introduce distributed models of computing. This entails computing on a network under various assumptions. We will present synchronous and asynchronous models and examine a basic task, breadth first search, in each.

2 Distributed Models.

The model considers computing on a network itself. That is, we are given a graph $G = (V, E)$, where one can send messages along the network. In a synchronous network, where the action of the network proceeds acording to a global clock. That is, in each step a message can be sent along each edge adjacent to a processor. Variants include restrictions on being able to send only a single message per step.

In this model, one can compute a breadth first search tree in $O(D)$ time with $O(V)$ messages as follows. From the root, you send a message labelled 0 to all the neigbors which mark themselves with distance 1. For each previously unmarked node one can then send a message to all their neighbors marked with 2. The algorithm continues with each node acting as follows: "If I am unmarked and I receive a message with label, l, I send out a message of label $l + 1$ on all adjacent edges."

The labels of the nodes give the depth of a node, the edge that marked the node is the parent edge in the breadth first search tree. The time before every one is marked is D where D is the depth of the tree, and the number of messages is $|E| + |V|$ since one sends at most one message per edge (if one is a bit careful) plus an extra message per tree edge.

3 An Asynchronous Model

Networks, however, may not typically come with a global clock. Here, we consider an asynchronous model. In this case, we do not assume that messages are delivered in a single step. Here, we assume that some may be delivered much faster. This introduces complications to the algorithm above. For example, if we run the synchronous algorithm above, we may label a node with a much higher label than its true depth in the breadth first search tree.

We could modify it slightly as follows: 'If I receive a message with label, $l < l'$, where l' is my previous lowest label (or infinite if I have not seen a previoius label') I send out a message of label $l + 1$ on all adjacent edges."

This algorithm computes the breadth first search. What is its complexity? In this model, we define the time to be the maximum "depth" of the computation. Or perhaps, by the normalized time. That is, we assume that no message takes longer than 1 unit of time, and then look at the time that it takes for the algorithm to complete. With this view, we see that the time of the asynchronous algorithm is once again, D. The message complexity is the *maximum* over all timings of the number of messages sent. Here, we can bound the number of messages by $O(|V||E|)$, since the label of a node can be update at most V times and each time it sends a message along each edge, totalling to $2|E||V|$.

Question 1 Give an example network and timing that requires $\Omega(|V||E|)$ messages.

Oops. This is much worse than the synchronous case. Indeed, a modification of the algorithm for shortest path, requires an exponential number of messages.

4 Taming the wild west

We can alleviate the message problem a bit by simulating the synchronous algorithm. How could one do that? Well, we are building a tree. We could use that tree to synchronize the process.

That is, in each step of the synchronous algorithm we extend the breadth first tree by one level. We can enforce this as follows using the following asynchronous algorithm.

Assume that at some stage there is a tree of depth d and the frontier nodes are the nodes at level d. (Initially, we have a trivial depth 0 tree.) At each stage, the root sends a go message to its children which forward this message to its children, until they reach the frontier nodes. When the frontier nodes, receive the go message, they send a $d+1$ message along all their edges. Each frontier node waits for a response along each of these edges and when it receives one, it reports back to its parent that it is done, and marks itself as an interior node (it is no longer on the frontier). Each interior node, waits to receive a done message from each of its children, and reports to its parent when it is done.

Finally, if an unmarked node receives a message with label l, it defines itself as a frontier node and responds to the sending node that it is its child (choosing arbitrarily) and responds to all other incoming messages that it is not.

This mimics the synchronous algorithm but requires $O(D|V|)$ additional messages for synchronization. The explore messages from the frontier nodes is $O(|E|)$ as with the synchronous algorithm. The time of this algorithm is $O(D^2)$.

That is the message complexity is $O(|E| + D|V|)$ and the time is $O(D^2)$. Thus, we have done better on message complexity but worse on time.

5 Local synchronization

The synchronization allowed us to control the messages to some extent, but cost us in time. We can trade this off a bit, by not forcing complete synchronization as follows.

We divide the process into D/d stages. In each stage, we extend our frontier by d levels. To do so, we run the algorithm of the previous section for depth d. This, gives a message complexity of $O(|E|d + |V|D)$ messages and a time complexity of D/d stages each of time $d^2 + D$. By choosing $d = \sqrt{D}$, we obtain an algorithm of message complexity $O(|E|\sqrt{D})$ and time complexity of $O(D^{1.5})$.

6 Local, nonoverlapping synchronization

It would be nice to synchronize the roots of a strip so that they do not flood the edges with messages. We do so, using the notion of cover trees of small diameter of the root nodes of a strip.

That is, we would like to have the following structure.

We define a strip cover consists of a set of trees T, that span the source nodes of the strip. The set of clusters of the cover are the source nodes in each tree. We define the *depth factor,* α of the strip cover as the maximum depth of any tree divided by d. The *load factor* λ is the maximum number of clusters within distance d of any node in the strip.

Now, given a strip cover of a strip. One can implement the stage for this strip in time $O(d^2\lambda)$ and with $O(|E_d|\alpha)$ messages. (This can be compared to $O(d^2)$ time and $O(|E_d|d)$ messages in the algorithm of the previous section.)

Thus, if α can be made sufficiently small (and one could find the strip cover in a distributed fashion), one could choose d to be much smaller than $\sqrt{D}$.

But, it is not immediately clear such an object exists. Well, not to me anyway. But, it does.

In particular, consider the following algorithm. Start with each source node, by itself, compute the load factor for each node in the strip. If it is less than α, we are done. If not there is a tree of depth $2d$ that contains α of the source nodes. We do this simultaneously, by building a graph among the source nodes, with an edge between two when they both hit a node in the strip of high load. We choose an independent set of such source nodes, and build trees by assigning other source nodes arbitrarily to one of their neighbors in the independent set.

Each tree, now contains at least α nodes. This takes a bit more argument.

This can be made into an induction that ensures that at stage i, the depth of the trees is at most $2^i d$ and the size of each tree is at least α^i.

By choosing α to be V^ϵ, this ensures that the depth is not more than $2^{1/\epsilon}d$. In particular, one can choose ϵ so that the depth factor and the load factor is at most

$$V^{1/\sqrt{\log N}}$$

.

Given such a cover, one can implement a strip bfs stage with much fewer messages.

How does one compute such a cover. Well, we could run the bfs algorithm from each node, with the caveat that each strip node stops participating after it gets the load factor α messages. The trees, in which it participates, in some sense defines the graph, we defined above. We then need to find an independent set "locally".

This, we can do by performing the algorithm for finding a maximal independent set on a PRAM. That algorithm only communicated among edges of the graph. That allows us to find a large independent set in $O(\log n)$ steps of communication along the edges. Each such communication requires only time that is proportional to the depth of the tree. Thus, one can find a strip cover in time that is linear in d and polynomial in the depth factor. Moreover, the number of messages is also polynomial in the depth factor and the load factor.

Now, we can design a breadth first search algorithm that proceeds recursively from large d. There is a rather complicated recursion, but we leave it to the very interested reader.