FINDING A MINIMUM SPANNING TREE
CAN BE HARDER THAN
FINDING A SPANNING TREE IN DISTRIBUTED NETWORKS

by

E. Korach, S. Moran and S. Zaks

Technical Report #294

October 1983
Finding a Minimum Spanning Tree Can be Harder than Finding a Spanning Tree in Distributed Networks

E. Korach

IBM Scientific Center
Technion City
Haifa, Israel 32000

S. Moran and S. Zaks

Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel 32000

ABSTRACT

We show that when the underlying graph of a network is the complete graph on \( n \) vertices, the problem of distributively finding a minimum spanning tree has a lower bound of \( O(n^2) \) messages. In the proof, we use a min-max equality for complete graphs, proved and evaluated here. This lower bound, together with the known upper bound of \( O(n\log n) \) messages for distributively finding a spanning tree in such a network, proves that finding a minimum spanning tree is sometimes harder than finding a spanning tree in a distributed network.
1. INTRODUCTION

The model under investigation is a network of \( n \) processors with distinct identities. Each processor has some communication lines, connecting him to some others; he knows the (non-negative) cost associated with each such line, but not the identities of his neighbors. The communication is done by sending messages along the communication lines. The processors all perform the same algorithm, that includes operations of (1) sending a message to a neighbor, (2) receiving a message from a neighbor and (3) processing information in their (local) memory.

We assume that the messages arrive, with no error, in a finite time, and are kept in order in a queue until processed. We also assume that any non-empty set of processors may start the algorithm; a processor that is not a starter remains asleep until a message reaches him.

The communication network is viewed as an undirected graph \( G = (V, E) \) with \( |V| = n \), and we assume that the graph \( G \) is connected. We refer to algorithms for a given network as algorithms acting on the underlying graph. An edge in this graph is considered unused during a certain execution of an algorithm as long as no message was sent along it (otherwise it is used).

Working within this model, when no processor knows the value of \( n \), a spanning tree is found in [1] in \( O(n \log n + |E|) \) messages for a general graph. It is pointed out in [1] that for general graphs no algorithm to find a spanning tree (ST) which is more efficient than their algorithm to find a minimum spanning tree (MST) is known.

We show here that when the underlying graph is complete, the problem of finding an MST is strictly harder than the one of finding an ST. More precisely, we show that any algorithm for finding an MST on a complete graph must require, in the worst case, \( O(n^2) \) messages, while it was shown in [2] that \( O(n \log n) \) messages are always sufficient (and - in the worst case - necessary) for constructing an ST in a complete network. Note that an algorithm that finds an MST in \( O(n^2) \)
messages is easily constructed.

2. MAIN RESULT

Our main result is the following theorem:

**Theorem:** Any distributed algorithm to find an MST on a complete weighted graph \( G = (V,E) \) with \( n \) vertices uses, in the worst case, \( O(n^2) \) edges of the graph.

**Proof:** We make use of the following basic property of distributed algorithms in the above model: A vertex cannot distinguish between unused edges of equal costs adjacent to himself (since he does not know the identities of the neighbors across these edges). Therefore, the proof can be described as a game between the network and an outside adversary performed on the graph \( G \). Whenever a vertex wants to send a message along an unused edge of a certain cost, the adversary chooses such an edge for him (if more than one exists). The goal of the adversary is to force the algorithm to use as many edges as possible before completion. Without loss of generality we assume that the algorithm must send messages along the edges of an MST before it halts. The adversary may exchange costs of unused edges as long as the sets of costs of the edges adjacent to each vertex do not change. For example, if we have as part of the graph the subgraph shown in Figure 1a, where the shown edges are unused, then the adversary can change the costs of the edges to these shown in Figure 1b; no vertex should be able to tell that any change was made in the network, since his local view of the network remained unchanged. This entails that the algorithm can now complete its task if and only if it could have done it before this exchange was made.
We take $G$ to be the complete graph of vertex set $V = X \cup Y$, where $X \cap Y = \emptyset$ and $|X| = |Y| = n \geq 2$. At the beginning all the edges within $X$ and within $Y$ have a cost of 0, and all the edges connecting $X$ and $Y$ have a cost of 1. It is important to note that this initial structure of the graph $G$ is known only to the adversary; as far as each single vertex is concerned, he only knows that there are $n$ edges of cost equal 1 and $n - 1$ edges of cost equal 0 adjacent to himself (and he can find
the identity of the vertex across an edge only by using this edge*).

The adversary can exchange costs of edges as described above. In fact, the adversary will make only one exchange of costs of the type depicted in Figure 1 (where \( x_1, x_2 \in X \) and \( y_1, y_2 \in Y \)) during the execution of the algorithm, whenever he chooses. Therefore the resulting MST will have a cost equal 0, and will use the edge \((x_1, y_1)\) or the edge \((x_2, y_2)\) or both.

When the algorithm starts, some vertex eventually asks for an (unused) edge of cost 0 or 1, and the adversary chooses such an edge at random from the initial structure above and gives it to this vertex. This process is repeated each time a vertex asks for an unused edge. When the algorithm stops the set of unused edges does not contain a cycle of four edges like the one shown in Figure 1a, called \( XY\)-square (where \( x_1, x_2 \in X \) and \( y_1, y_2 \in Y \)). If it does, then the MST produced by it uses at least one edge of cost 1 (connecting \( X \) and \( Y \)), and by making at this point an exchange as described above the adversary creates an ST of cost 0, which proves the algorithm to be incorrect. Thus, the algorithm must use a set of edges that intersects all such cycles before it stops. As we show in the next Lemma, such a set contains at least \( n \choose 2 \) edges. This will complete the proof of the Theorem.

Let \( \nu \) denote the maximum number of \( XY\)-squares which are pairwise edge disjoint, and \( \tau \) denote the minimum number of edges that meet every \( XY\)-square.

**Lemma:** Let \( G = (V,E) \) be a complete graph on \( 2n \) vertices where \( V = X \cup Y \), \( X = \{x_1, x_2, \ldots, x_n\} \) and \( Y = \{y_1, y_2, \ldots, y_n\}, n \geq 2 \). Then

(i) \( \nu = \tau = \frac{n}{2} \)

(ii) \( E(X) \) and \( E(Y) \) are the only sets of cardinality \( \tau \) that meet all \( XY\)-squares.

* or by using all his other adjacent edges.
Proof:

(i) Clearly $E(X) \geq \tau \geq \nu$. It is easy to see that $S$ is a collection of pairwise edge disjoint $XY$-squares, where:

$$S = \{(x_i, x_j), (x_j, y_i), (y_i, y_j), (y_j, x_i) : (x_i, x_j) \in E(X)\}$$

The cardinality of $S$ is $|E(X)|$ which implies that $\nu \geq |E(X)|$. We have

$$|E(X)| \geq \tau \geq \nu \geq |E(X)| = \left[\frac{n}{2}\right].$$

It follows that $\nu = \tau = \left[\frac{n}{2}\right]$. 

(ii) Let $T$ be a minimum set of edges that meets all $XY$-squares (i.e. $|T| = \tau$). If $T$ contains an edge $e$ from $X$ to $Y$, relabel $X$ and $Y$ (if necessary) so that $e = (x_1, y_1)$. Clearly $e$ does not meet any $XY$-square in $S$, hence $|T| > \left[\frac{n}{2}\right]$, a contradiction. If $T$ meets both $E(X)$ and $E(Y)$ then (since $|E(X)| = |E(Y)| = |T|$) we have an edge in $E(X)$ and an edge in $E(Y)$ both not in $T$. It follows that these two edges are in an $XY$-square that does not meet $T$. This implies (ii).

Q.E.D.

Acknowledgement: We would like to thank Reuven Bar-Yehuda for some helpful comments.
REFERENCES
