

Minimum Spanning Tree

Given a weighted graph $G = (V, E, w)$, goal is to compute the MST for G , i.e., a spanning tree of G whose weight is minimum.

Weights can represent latency, bandwidth, traffic load etc.

Time Complexity model: We will assume that it takes unit time to send a message of size $O(\log n)$ across an edge.

Goal: At the end of the distributed MST algorithm, each node will know which of its neighbors belong to the MST and which do not.

Can be easily modified to produce a rooted tree.

W.l.o.g., assume that all edge weights are distinct. This implies a unique MST.

MST Basics

Let T_M be the (unique MST) of G .

A **(MST) fragment** is a subtree of T_M .

An *outgoing edge* of a fragment is an edge in E where one adjacent node to the edge is in the fragment and the other is not.

The minimum weight outgoing edge (MOE) of fragment T is called $M(T)$.

The following rule produces a MST.

Blue rule: Pick a fragment T and add $M(T)$ to it (until possible).

Synchronous GHS Algorithm

The algorithm starts with each individual node as a fragment by itself and ends with one fragment — the MST.

Initially, each node (a singleton fragment) is a root node; subsequently each fragment will have one root node.

Each nodes in the fragment will know its parent and children.

Each fragment is identified by the identifier of its root and each node knows this.

One phase of GHS

All fragments find their MOE simultaneously in parallel.

To find the MOE of a fragment, the root in the fragment broadcasts a message to all nodes in the fragment using the edges in the fragment.

TO find the minimum weight outgoing edge incident on itself, a node checks its neighbors in increasing order of weight.

Each node in the fragment, after receiving the message, finds the minimum outgoing edge adjacent to it and reports to the root by convergecast.

Once the MOE of the fragment is found, the fragment attempts to combine with the fragment at the other end of the edge.

If the other fragment has also selected the same edge as its MOE then the two fragments agree to combine at that point. The endpoint of this edge with the higher identifier becomes the root of the combined fragment.

The (combined) root broadcasts a "new-fragment" message through the fragment edges and the MOE edges.

Each node updates its parent, children, root identifier.

Analysis

$O(\log n)$ phases.

In each phase takes $O(n)$ time.

Hence, time complexity is $O(n \log n)$.

Each phase takes $O(n)$ messages (for convergecast and broadcast) plus the messages needed to find MOE.

The latter takes a total of $O(|E|)$ for the entire algorithm, because, a node need not checks a neighbor at most once.

Hence total message complexity is $O(|E| + n \log n)$ which is optimal.

Asynchronous Version

Each fragment has a level.

A fragment containing only a single node is at level 0. Let the fragment F be at level L , the edge e be the MOE of F , and the fragment F' be at the other end of e . Let L' be the level of F' . We have the following rules for combining fragments:

(1) If $L < L'$, fragment F and F' are combined into a new fragment at level L' , and the root node of F' is the root node of the new fragment.

(2) If $L = L'$ and fragments F and F' have the same minimum-weight outgoing edge, F and F' are combined into a new fragment at level $L + 1$. The node incident on the combining edge with the higher identity number is the root node of the new fragment.

(3) Otherwise, fragment F waits until fragment F' reaches a level high enough to apply any of the above two rules of combining.

The waiting of the fragments in the above procedure cannot cause a deadlock.

The waiting is done to reduce the communication cost (number of messages) required for a fragment to find its MOE.

The communication cost is proportional to the fragment size, and thus communication is reduced by small fragments joining into large ones rather than vice versa.

The maximum level a fragment can reach is $\lg n$, where $n = |V|$.

The algorithm takes $O(n \lg n + |E|)$ messages and $O(n \lg n)$ time.