Minimum Spanning Tree Construction

Arvind Krishnamurthy
Fall 2003

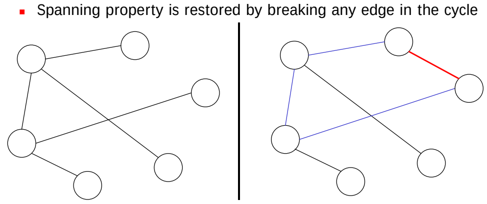
Faster Leader Election in General Networks

- General approach:
  - Build a spanning tree of the entire network
    - Each node determines a parent
  - Root of the tree is the leader

- In fact, compute not just any spanning tree:
  - Compute the “minimum spanning tree” in the network
  - Assumes that channels have some kind of “weight” or “cost” that needs to be minimized
  - Useful for determining an “efficient” subgraph over which communication can take place

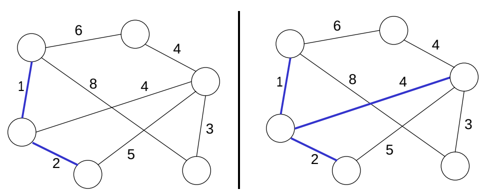
Basic facts of MST

- MST Properties:
  - Spanning tree: includes all “n” vertices
    - Number of edges in tree: n-1
    - Contains no cycles
  - Minimum weight amongst all spanning trees
  - When we an edge “e” to a spanning tree, a cycle is created
  - Spanning property is restored by breaking any edge in the cycle



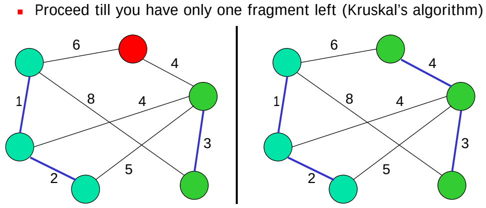
MST Construction

- Let T be a fragment of the MST
  - Spans only a subset of the vertices
  - Method to extend T
    - Find some edge that has only one endpoint in vertices of T
    - Find such an edge with the minimum weight
    - T can be extended by including the edge
- Prim-Dijkstra: start with some vertex and extend the tree



MST Construction (contd.)

- Alternately:
  - Start with “n” vertices being “n” different fragments
  - Consider the minimum weight edge connecting two fragments
  - Include the edge and combine the corresponding fragments
    - Since the edge connects two fragments, its addition cannot create a cycle
  - Proceed till you have only one fragment left (Kruskal’s algorithm)



Distributed Algorithm

- Assume that there is a separate process associated with each vertex
  - Assume that it is connected to a few other processes
  - Each process knows the weights attached to its links

- Would like to maximize concurrency
- Guarantee safety properties:
  - No cycle at any point in the algorithm

- How do we design such a distributed algorithm?

First Attempt

- Let's develop a concurrent version of Kruskal's algorithm
- Start with "n" fragments:
  - Each fragment consists of just one process/node
  - Each node is the leader of its fragment
  - Each node's "parent" is itself

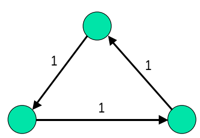
1) Have each fragment determine:
   - An outgoing edge (connects to another fragment)
   - A minimum weight edge
   - Referred to as the minimum-weight outgoing edge (MWOE)
2) Combine the two fragments
3) Elect some element from the combined fragment as the leader
4) Patch up the parent links to point towards the leader

Discussion

- Is our approach feasible?
  - Can we implement all the steps of our algorithm?
  - Can we guarantee the safety property during each step of the execution?

Problem: creation of cycles

- Each fragment finds a MWOE
- Connects to another fragment along the MWOE
- Cycles could be created
- Simple example:



- How do we fix this problem?

Solution to cycle creation

- Guarantee that each edge in the system has an unique weight
- Even if edges do not have unique weights, can make them unique
  - Take the edge cost and append the node uids to the edge cost
  - Weight is now a triple: actual edge cost, node1-uid, node2-uid
  - Comparing weights:
    - First compare edge costs
    - If same, compare node1-uids
    - If same, compare node2-uids

Algorithm Exposition

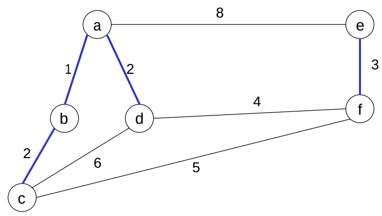
1) Each fragment finds MWOE
2) Combine fragments using MWOEs
3) Elect some element from the combined fragments as the leader
4) Patch up the parent links to point towards the leader

- How do we determine whether an edge is "outgoing" or not?

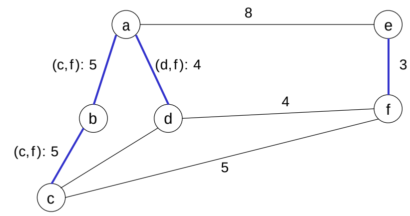
Detailed Algorithm

Step 1: Each fragment finds MWOE
- Leader initiates search for MWOE
- Sends a broadcast message through the fragment's tree links
- When the search is received, each node knows who is the parent
  - Patching parent links (step 4) is accomplished
- When the search is received, each node explores its links:
  - Finds whether it is outgoing or not
  - Uses the leader of the other fragment to determine this fact
  - Queries the node on the other side of the edge for its leader
    - Are there any pitfalls in this approach?
- Each node finds its MWOE and reports it to its parent
- A convergecast operation takes place to find the MWOE of the fragment

Algorithm (contd.)



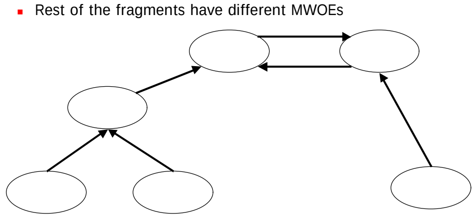
Algorithm (contd.)



- Once the leader determines the MWOE of the entire component:
  - It sends a connect message across the MWOE
  - Marks the edge as a “branch” of the spanning tree

Cascading Connections

- Multiple fragments could connect with each other at the same time
- If “k” fragments combine:
  - Two fragments share a MWOE
  - Rest of the fragments have different MWOEs



Electing new leader

- Consider the two nodes on either side of the common MWOE
  - Elect one of them as the leader of the combined fragment
  - Say, choose the one with the lower uid

- Algorithm:
  1) Each fragment finds MWOE:
     1) Broadcast “search” along all “branch” edges, do not send to parent
     2) Test for outgoingness
     3) Convergecast
  2) Combine fragments using MWOEs
     1) Leader sends a “connect” message across MWOE, marks edge as a branch edge
  3) Elect some element from the combined fragments as the leader
     1) Pick common MWOE and choose one of the two endpoints
  4) Patch up the parent links to point towards the leader
     1) Included in step 1.1

Testing for “outgoingness”

- Requires each node to know who is the fragment’s leader
- Fragment’s leader is determined during the search broadcast
  - Along with the search command, also send leader’s identity
- Some nodes could receive the broadcast search before other nodes
  - Some nodes have current leader information, other nodes might not
- Solve this problem first in “synchronous”, “non-uniform” setting
  - Assume that algorithm unfolds in levels
  - Make sure that “n” steps are spent before progressing to next level
  - When “outgoingness” test is made, wait for the level number to increase to current level on the target node

Synchronous Algorithm

1) Initialize count to zero
2) Each fragment finds MWOE:
   1) Broadcast “search” along all “branch” edges, do not send to parent
   2) Test for outgoingness
   3) Convergecast
3) Combine fragments using MWOEs
   1) Leader sends a “connect” message across MWOE, marks edge as a branch edge
4) Elect some element from the combined fragments as the leader
   1) Pick common MWOE and choose one of the two endpoints
5) Wait for count to reach “n”, repeat process from step (1)

What is the message complexity and time complexity?
   How many levels does the algorithm go through?
   At each level, how many messages are being sent?

Announcements

- Some more sample C-code using TCP on Zoo machines:
  - Code is at /c/cs425/socketv2
  - Code is unithreaded version that monitors multiple sockets
- Pthreads tutorial:
  http://www.cs.nmsu.edu/~jcook/Tools/pthreads/pthreads.html