Econ 172A - Slides from Lecture 14

Joel Sobel

November 15, 2012

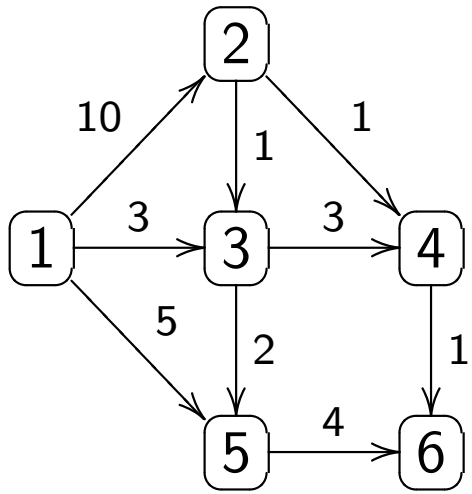
Announcements

- Midterm: Graded. Information on Webpage.
- If you have concerns about grading, pay attention to instructions on regrades. In particular, do not write on the exam.
- I will have no office hours next Tuesday (November 20).
- Quiz 3: Today. Please stay in your seat after you finish.
- Problems on Branch and Bound: 2007: Homework 3, #1; Final: #5,6 2008: Homework 3 # 1, 2; Final: #4.
- Relevant Homework Problems (Network Models): 2008, Problem Set 3, 2-4. Final: 1-3, 6.
  2007, Problem Set 3, 4-5. Final: 1-4, 7.

Example

The figure (next page) represents a railroad network. The numbers beside each arc represent the time that it takes to move across the arc. Three locomotives are needed at point 6. There are two locomotives at point 2 and one at point 1. We can use the shortest route algorithm to find the routes that minimize the total time needed to move these locomotives to point 6.

The Network



Tuesday we solve the problem of finding the shortest route from (1) to (6). Now we find the shortest route from (2) to (6). To use the algorithm, write temporary labels is the following table.

It is not possible to pass through (1).

Iteration	2	3	4	5	6
1	0^{*}	1^{**}	1^{**}	∞	∞
2	0^{*}	1^{*}	1^{*}	3	2^{**}
3	0^{*}	1^{*}	1^{*}	3^{**}	$2*$

- In the first iteration, we can put a permanent label on both Node (3) and Node (4) since both are minimum cost temporary labels.
- From the table we see that the shortest distance from (2) to (6) is 2.
- The shortest route is (2) $\rightarrow$ (4) $\rightarrow$ (6).
- Since we have two locomotives at Node (2) and one at Node (1), the total distance is $7 + 4 = 11$.

Animation

Here is a link to a webpage that demonstrates how the algorithm works:
`Shortest Route`

Theory

- Algorithm stops in a finite number of iterations.
- Why? Each step you get at least one more permanent label.
- A permanent label is the cost of a shortest route.
- At Iteration 1 this is obvious.
- For all future iterations, it is true.
- The newest permanent label (say at Location (N)) gives a cost that is lower than any other route through an existing temporary label.

Because . . .

If this label does not represent the cost of a shortest route, then the true shortest route to Location (N) must pass through a node that does not yet have a permanent label – at a cost greater than that of the minimum temporary label – before reaching Location (N). Since all cost are nonnegative, this route must cost at least as much as the temporary label at Location (N).

Formulation

Given a shortest route problem with costs $c(i,j)$, Node (0) is the source and Node (N) is the sink. Let $x(i,j) = 1$ if the path goes from i directly to j and 0 otherwise. The following problem describes the shortest route problem:
Find $x(i,j)$ to solve:

$$\min \sum_{i=0}^{N} \sum_{j=0}^{N} c(i,j)x(i,j)$$

subject to:

1. $\sum_{i=0}^{N} x(i,j) = \sum_{k=0}^{N} x(j,k)$ for $j = 1, \ldots N-1$,
2. $\sum_{k=1}^{N} x(0,k) = 1$,
3. $\sum_{i=0}^{N-1} x(i,N) = 1$, and $x(i,j)$ either zero or one.

1. (1) says that the number of paths leading to any Node (j) is equal to the number of paths leading out of the node.
2. (2) says that at exactly one of the edges in the path must come from the source.
3. (3) says that exactly one of the edges in the path must go to the sink.
4. The objective function sums up the cost associated with all of the edges in the path.
5. This formulation requires that all of the costs be non negative.

Minimal Spanning Tree Problem

- Each edge from Node (i) to Node (j) has a cost $(c(i,j))$ associated with it.
- The edge (i,j) represents a way of connecting Node (i) to Node (j).
- An arc could represent an underground cable that connects a pair of computers.
- Goal: find out a way to connect all of the nodes in a network in a way that minimizes total cost (the sum of the costs of all of the edges used).

Terminology

1. A **tree** is just a collection of edges that is connected and contains no cycles.
2. A collection of nodes is **connected** if you can go from any node to any other node using edges in the collection.
3. A **cycle** is a collection of edges that starts and ends at the same node.
4. A **spanning tree** is a tree that contains all of the nodes in a network.
5. A **minimal spanning tree** is a spanning tree that has the smallest cost among all spanning trees.

Algorithm

Step I Start at any node. Join that node to its closest (least cost) neighbor. Put the connecting node into the minimal spanning tree. Call the connected nodes C.

Step II Pick a node j^* that is not in C that is closest to C (that is, Node (j^*) solves: $\min\{c(i,j) : i \in C, j \notin C\}$). Put Node $(j*)$ into C and put the cheapest edge that connects C to $j*$ into the minimal spanning tree.

Step III Repeat this process until a spanning tree is found. If there are N nodes in the network, then the minimal spanning tree with have $N-1$ edges.

Ties

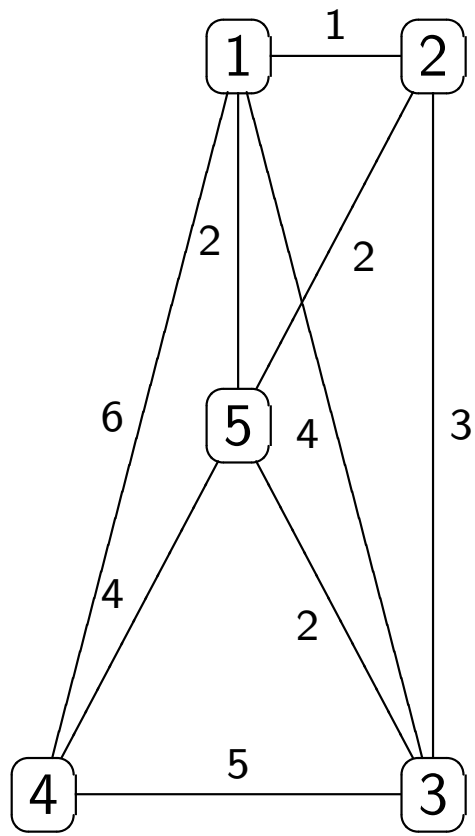
Don't worry. Break them arbitrarily.

Greedy Algorithm

- The algorithm is called **greedy** because it operates by doing the cheapest thing at each step.
- This myopic strategy does not work to solve all problems.
- I will show that it does work (to find the minimum cost spanning tree) for this particular problem.

Example

To see how the algorithm works, consider the following example. Suppose that five social science departments on campus each have their own computer and that the campus wishes to have these machines all connected through direct cables. Suppose that the costs of making the connections are given in the network, below. (An omitted edge means that a direct connection is not feasible.) The minimum spanning tree will determine the minimum length of cable needed to complete the connections.

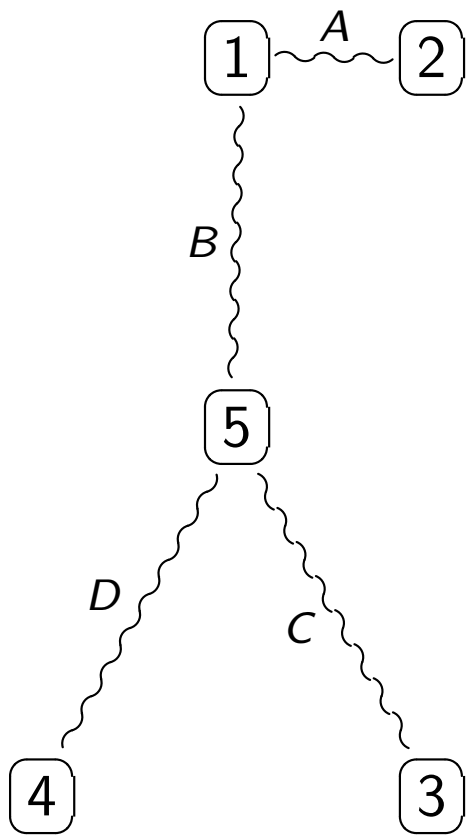

1
2
3
4
5
1
2
2
6
4
3
4
2
5

Computation

1. Start at Node (1).
2. Add Node (2) into the set of connected nodes (because $c(1,2)$ is smaller than $c(1,j)$ for all other j.
3. This creates the first edge in the minimal spanning tree, which a labeled A below.
4. The minimum cost of connecting Node (3) or Node (4) or Node (5) to Node (1) or Node (2) is two.
5. This minimum is attained when Node (5) is added.
6. Connect Node (5) next and add either $(1,5)$ or $(2,5)$ into the tree.
7. I added (1.5) and called this edge B.
8. The third step connects Node (3) and adds the edge $(3,5)$.
9. The final step connects Node (4) and adds the edge (4.5).

Network

- The letters (in alphabetical order) indicate the order in which new edges were added to the tree.
- The total cost of the tree is nine.



Theory

1. Denote by N_t the nodes connected in the t^{th} of the algorithm and by N_t' all of the other nodes.
2. I want to show that each edge added to the tree is part of a minimal spanning tree.
3. Let S be a minimal spanning tree.
4. If the algorithm does not generate S, then there must be a first step at which it fails.
5. Call this step n.
6. The edges identified in the first $n-1$ steps of the algorithm are part of S, but the edges identified in the first n steps are not.
7. Take the edge added at the n^{th} step of the algorithm.
8. This edge connects N_n to N_n'.
9. Put it into the tree in place of the edge connecting N_n to N_n' in S.
10. This leads to a new spanning tree S', which must also be minimal (by construction, the algorithm picks the cheapest

Another Algorithm

1. At each stage of the algorithm you pick the cheapest available branch (if there is a tie, break it arbitrarily), provided that adding this branch to your existing tree does not form a cycle.
2. If adding the cheapest branch does create a cycle, then do not add that branch and move on to the next cheapest.

Comments

- Both algorithms are "greedy" in the sense that they work by doing the myopically optimal thing without looking for future implications.
- Problems that can be solved using such an algorithm are combinatorially easy (in the sense that it is computationally feasible to solve large versions of the problem).
- Not all problems can be solved by a greedy algorithm and the proof that a greedy algorithm works is not always simple.

Remember Coin changing

Remember Knapsack problem

Scheduling

Job j	Deadline d_j	Penalty w_j
1	1	10
2	1	9
3	3	7
4	2	6
5	3	4
6	6	2

- A number of jobs are to be processed by a single machine.
- All jobs require the processing time of one hour.
- Each job j has a deadline d_j and a penalty w_j that must be paid if the job is not completed by its deadline.

Greedy Algorithm

- Choose the jobs one at a time in order of penalties.
- Largest first.
- Reject a job only if its choice would mean that it, or one of the jobs already chosen, cannot be completed on time.
- The greedy algorithm tells you to do job one (highest penalty); to skip job two (if you do job one you will never finish job two on time); and then to do jobs three and four.

Solution

The algorithm tells you to do jobs 1, 4, 3, and 6 in that order.

Theory

Omitted. Note that it is easier to implement the algorithm than to prove that it works.