Maximum Flow

Introduction

Maximum flow is a subtopic in mathematical optimization, which uses math to select the best(most efficient) sets of elements in all the alternatives. These types of problems are usually given in a map, which contains several nodes, edges connecting each nodes, a flow value denoting to the limit for number of things that can go through this edge at once (weight), a starting node(source) and a destination node(sink). Then the problems asks for the maximum amount of a quantity that can be sent to the destination without exceeding the limit of each edge. A more human relatable example would be this:

"A company owns a factory located in city X, where products are manufactured. These products need to be transported to the distribution center in city Y. You are given the one-way roads that connect pairs of cities in the country, and the maximum number of trucks that can drive along each road. What is the maximum number of trucks that the company can send to the distribution center?"

This type of problem is easy if there are only a few nodes. Once the amount of nodes get larger , the problem becomes insolvable by humans, which means computers have to come in place.
Thankfully, MATLAB makes it all easy. It provides a function for solving these types of problems called "`graphmaxflow`". We will be looking at this function in this tutorial.

The syntax of this function is [`MaxFlow, FlowMatrix, Min-Cut] = graphmaxflow(Map, Starting Node, Destination Node)`.

Definitions

Definitions of Outputs

- FlowMatrix: a sparse matrix that records all the edges that go through and the flow value of those edges.
- MaxFlow: the sum of all the flow values from the chosen edges.
- Min-Cut: a vector which includes the nodes connected to the starting node after calculating the minimum cut. If there is more than one solution to the minimum cut problem, the cut will be a matrix.

Necessary Inputs

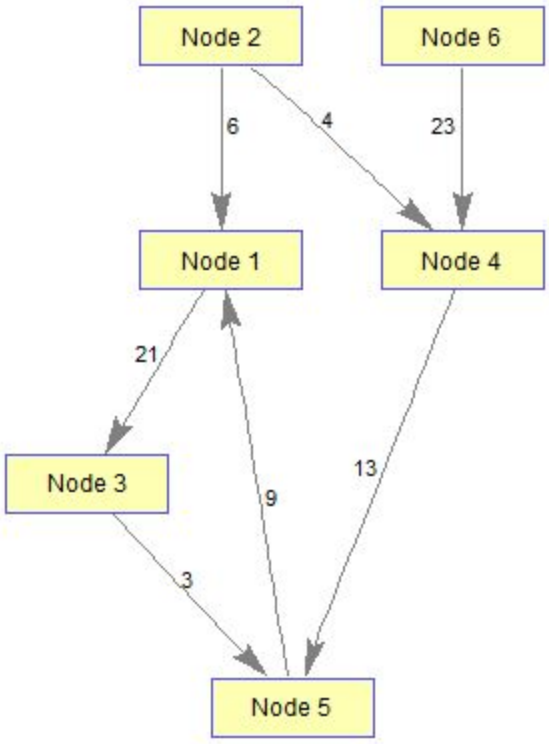
- Map is a sparse matrix including all the edges, their directions and flow values. The declaration is slightly different than that of the normal matrix. When creating a sparse matrix, you must use the `sparse`() function. It can be used in a lot of places and in a lot of different formats. For the use of map, the format is:
  ```
  map = sparse([First row, the starting node of the edge], [Second row, the ending node of the edge], [Third row, the flow value of the edge], number of nodes, number of nodes)
  ```
  Because it is a matrix, the length of each row must be the same. The length of each row is the number of edges in the map.

(Note: A cut is a set of edges. If these edges are removed from the map, there would not be any possible route to go from the starting node to the destination node. A minimum cut is the cut with the smallest sum of flow values out of all the possible cut. This is not the focus of this tutorial, so we will not go deeper into this.)

For example:
map = sparse([1, 3, 2, 6, 2, 4, 5], [3, 5, 1, 4, 4, 5, 1], [21, 3, 6, 23, 4, 13, 9], 6, 6)
which will generate 7 edges and 6 nodes with the following information:

Edge	Starting node	Ending node	Flow value
Edge 1	1	3	21
Edge 2	3	5	3
Edge 3	2	1	6
Edge 4	6	4	23
Edge 5	2	4	4
Edge 6	4	5	13
Edge 7	5	1	9

The map will look like this:



Note that each edge has a direction. This means going from node 1 to node 6 is impossible; however, going from node 6 to node 1 is possible, with edges 4, 6 and 7. Keep in mind that the map declaration is poorly written because the edges are not in order. It is best to organize them in order so it is easy to understand. A better organized version of the map would be:

```
map = sparse( [1, 2, 2, 3, 4, 5, 6], ...
              [3, 1, 4, 5, 5, 1, 5], ...
              [21, 6, 4, 3, 13, 9, 23], 6, 6)
```

(Note: Those "..."'s at the end of each line mean that the next line of the code is part of the command. If they were not added, matlab would think those were three different commands and therefore show an error.)

- Starting Node: This input only requires a single number. This number will indicate which node is the starting node. For example, with the same map, if the input is 6, then the starting node will be Node 6.
- Ending Node: This input only requires a single number, indicating which node is the destination node. For example, with the same map, if the input is 1, then the destination node will be Node 1.

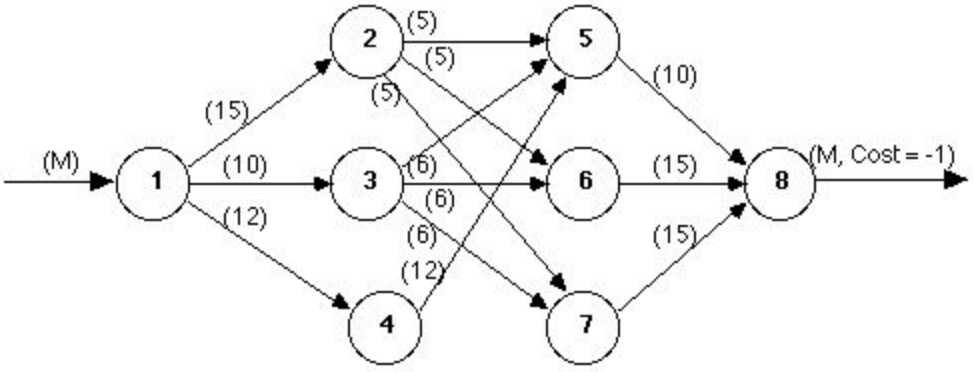
Name-Value Pairs

Setting	Value
'Capacity'	A column matrix sets the customized capacities of all the edges. The number of rows of this matrix must match that of the map. The order of edges are used to determine which value in this matrix goes to which edge. A better organized map will make the process of matching a lot easier.
'Method'	The algorithm used to find the minimum spanning trees (MST). The possible inputs are: 'Edmonds' - Edmonds and Karp algorithm. Time complexity is O(N*E^2). 'Goldberg' - the default algorithm. Goldberg algorithm. Time complexity is O(N^2*sqrt(E)). Note: N and E are the number of nodes and edges

A Sample Problem

This problem is found online. See the link below.
http://www.me.utexas.edu/~jensen/models/network/net11.html

The maximum flow problem is again structured on a network. Here the arc capacities, or upper bounds, that are relevant parameters.
The problem is to find the maximum flow possible from some given source node to a given sink node. A network model is in Fig. 17. All arc costs are zero, but the cost on the arc leaving the sink is set to -1. Since the goal of the optimization is to minimize cost, the maximum flow possible is delivered to the sink node.



First we have to set up the network, or map. To do that, we have to find the starting node, ending node and the flow value of every edge. This table contains all the edges and their information in an organized way:

Edge	Starting Node	Ending Node	Flow Value
Edge 1	1	2	15
Edge 2	1	3	10
Edge 3	1	4	12
Edge 4	2	5	5
Edge 5	2	6	5
Edge 6	2	7	5
Edge 7	3	5	6
Edge 8	3	6	6
Edge 9	3	7	6
Edge 10	4	5	12
Edge 11	5	8	10
Edge 12	6	8	15
Edge 13	7	8	15

Using those information, we can set up the map:

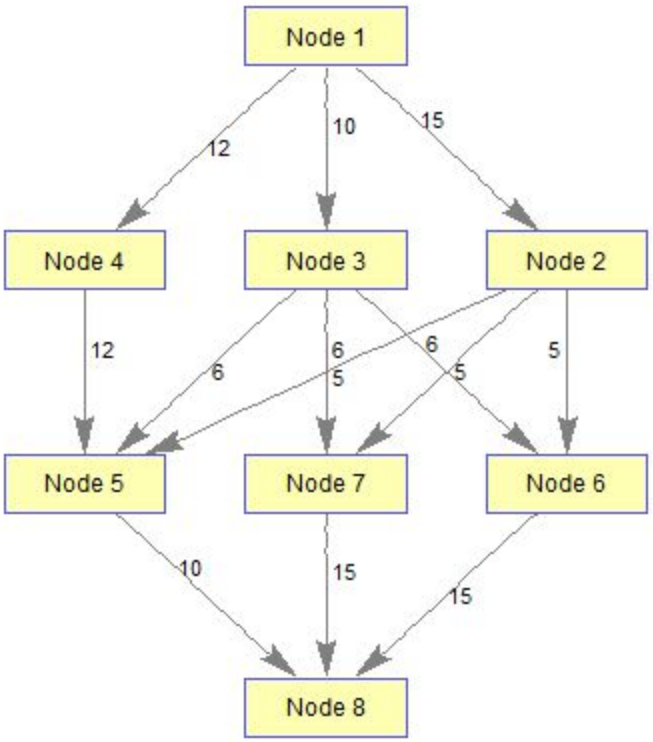
```
map = sparse( [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 5, 6, 7], ...
       [2, 3, 4, 5, 6, 7, 5, 6, 7,5,8, 8, 8], ...
      [15, 10, 12, 5, 5, 5, 6, 6, 6,12, 10, 15, 15], 8,8)
```

To check if the map is correct, you can graph it out using the following command:

```
view( biograph(map, [], 'ShowWeights','on'))
```

(Note:
biograph is a command to create a biograph object.
view is a function that opens a window and draw the biograph object.
map is the map that you want to draw.
[ ] is an array for nodes' IDs. It is left blank because we only want simple numbers for nodes' IDs in this case.
'ShowWeights' and 'on' are a name-value pair. It is used to set the visibility of flow values of each edge. The only inputs are 'on' and 'off'.
Visit http://www.mathworks.com/help/bioinfo/ref/biograph.html for more information)

Entering `view(biograph(map,[],'ShowWeights','on'))`, you will get an output that looks something like what is displayed below:



The above matches the graph that is provided.

The source (starting node) in the problem is Node 1, and the sink (ending node) is Node 8. When we plug these numbers into our function:

```
[maxFlow, maxFlowMap, minCut] = graphmaxflow(map, 1, 8)
```

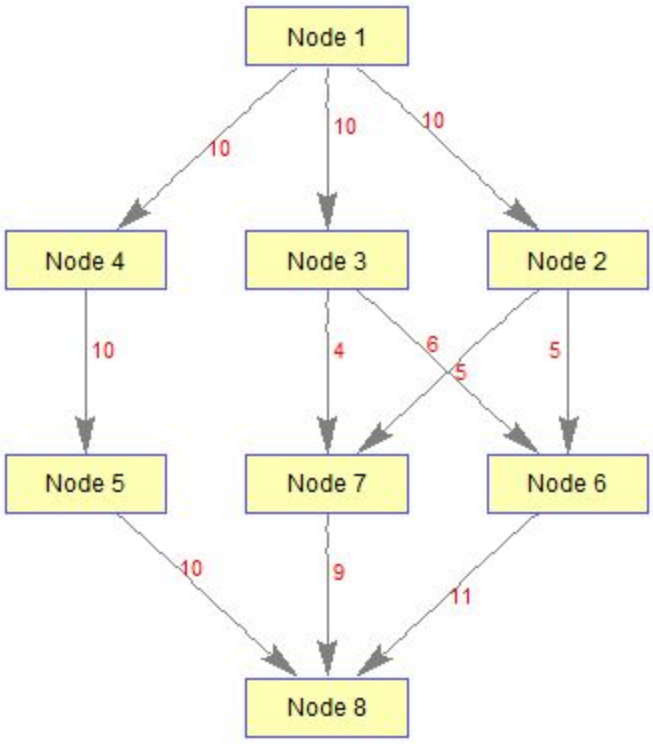
We get:

```
maxFlow =
     30

maxFlowMap =
   (1,2)       10.0000
   (1,3)       10.0000
   (1,4)       10.0000
   (4,5)       10.0000
   (2,6)        5.0000
   (3,6)        6.0000
   (2,7)        5.0000
   (3,7)        4.0000
   (5,8)       10.0000
   (6,8)       11.0000
   (7,8)        9.0000

minCut =
     1     1     0     1     1     0     0     0
```

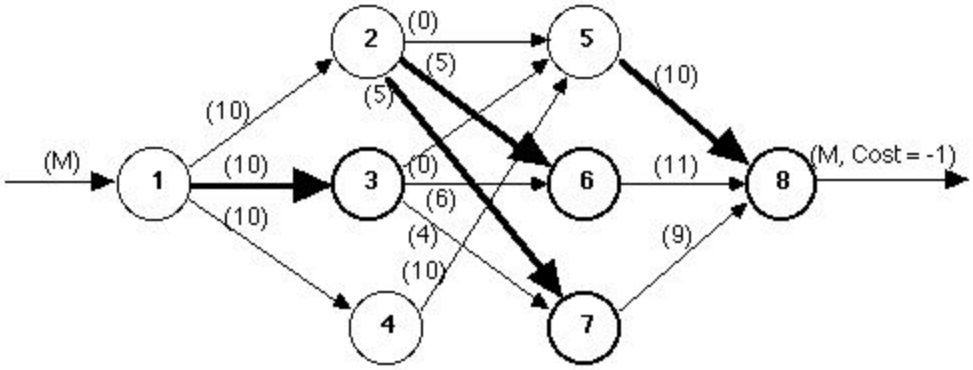
The answer given by the sample problem website is 30, which is exactly the same as ours.

To show the solution visually, draw the map again, but this time, change map to maxFlowMap.

```
view(biograph(maxFlowMap,[],'EdgeTextColor', [1,0,0],
'ShowWeights','on'))
```



Comparing to the answer they give:



Despite the difference in graphing styles, they are exactly the same.