CS 435 : Linear Optimization — Fall 2006

Lecture 32: Network Flows (contd.)

Lecturer: *Sundar Vishwanathan* — Scribe: *Ankit Jain*
Computer Science & Engineering — Indian Institute of Technology, Bombay

In this lecture we continue with the primal-dual algorithm for network flows.

1 Primal Dual Algorithm

The network flow problem as defined in the previous lecture is as follows :

$$max \sum_u f_{su}$$

$$\text{s.t. } \forall u, v \; f_{uv} \leq C_{uv} \tag{1}$$

$$\forall u, v \; f_{uv} + f_{vu} = 0 \tag{2}$$

$$\forall u \neq s, t \; \sum_v f_{uv} = 0 \tag{3}$$

where f_{uv} is the flow from vertex u to vertex v
and C_{uv} is the (non negative) capacity of the link u - v

In this case, we will solve the primal itself rather than the corresponding dual.

1.1 Initialization

We begin with a feasible solution.
$\forall u, v \; f_{uv} = 0$

Now, we need a piece of code that we can run again and again recursively, improving the above solution till we find the optimal. The Primal-Dual algorithm helps us find just that.

1.2 Recursion

The Primal Dual Algorithm

$$max \sum_u f'_{su}$$

$$\text{s.t. } \forall u, v \text{ where } (\; f_{uv} = C_{uv} \;) \; f'_{uv} \leq 0 \tag{4}$$

$$\forall u, v \; f'_{uv} + f'_{vu} = 0 \tag{5}$$

$$\forall u \neq s, t \; \sum_v f'_{uv} = 0 \tag{6}$$

This is related to the original problem by the following equation

$$\forall u, v \; f^{old}_{uv} + \epsilon f'_{uv} = f^{new}_{uv} \tag{7}$$

Our aim is to keep raising ϵ until we reach an equality of the form $f^{new}_{uv} = C_{uv}$

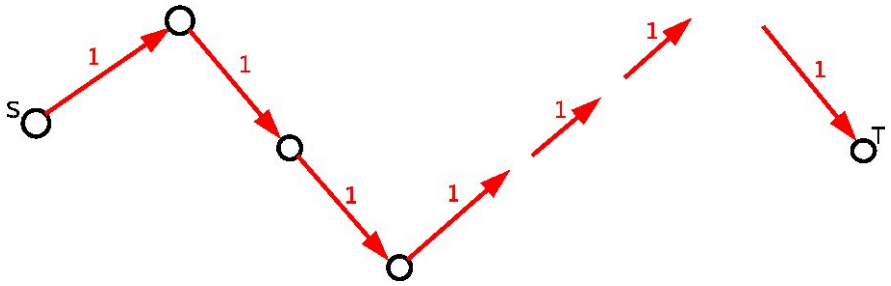

Figure 1: Path from s to t

$$\text{If } \sum_u f'_{su} = 1$$

the network flow must be of the form as shown in Figure1 since we have a net flow of 1 leaving from s which cannot disappear at any node except the destination.
All these edges have $f_{uv} < C_{uv}$

So all we need to do is to find one such path and raise the flow till one of the paths reaches its capacity. For any path, maximum raise possible $= C_{uv} - f_{uv}$ This tells us the new capacity if all the edges in the graph. So we construct a graph on same set of vertices with capacity $= C_{uv} - f_{uv}$

One interesting fact to note here is that the number of edges can infact increase on doing the above step. This can happen when we have a capacity of a edge = 0 and corresponding flow $(f_{uv}) < 0$. However there can only be a maximum of 2m edges. (where m is number of edges in input graph)

So given new capacities, we find any path from s to t. On the path, find the minimum capacity and keep ϵ = minimum capacity on that path. This will give us a new equality of the form $f_{uv} = C_{uv}$. Hence we have a required equality, and we move to the next recursion.

1.3 Order of Algorithm

If in Figure 2 we choose the path containing edge with weight 1 in each iteration it will take O(W) time to complete the execution. However the input size is O(ln W). (No of bytes required to represent total weight). So clearly the runtime is exponential in size of input. Hence choosing the appropriate path is critical. However LP cannot help us with this and we must look for a combinatorial algorithm. One such algorithm has been discussed in the next section.

1.4 Optimality

Another step of algorithm is to check if we have reached the optimal solution after each recursion. For this purpose, we look at all vertices accessible from s. We then seperate the graph into 2 parts, s on on side and t on the other. This is also know as the *cut*. For optimality we cannot have any flow through any cut. (The dual infact asks for the cut of minimum capacity.)

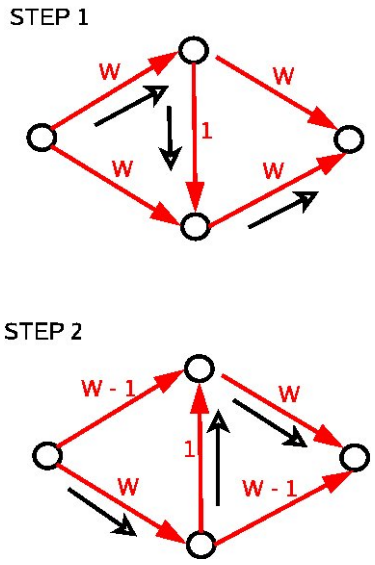

Figure 2: Example Graph

2 Improved Algorithm

2.1 Algorithm to find optimal path

The optimal path can be thought of as the path connecting s to t such that its minimum capacity edge is maximum among all paths from s to t. This can done with a simple modification to the Dijkstra's Algorithm (Tutorial 3 - Question 2).

2.2 Order of Algorithm

For calculating the order, we first make two important observations.

Observation 1

Let f be the current flow
and f^* be a flow of maximum value.

Claim : $f^* - f$ is a flow in the graph
Proof : Both f and f^* are flows in the original graph and hence must satisfy equations (1), (2) and (3). Therefore we have,

$$\forall u, v \; f^*_{uv} - f_{uv} \leq C_{uv} - f_{uv} \tag{8}$$
$$\forall u, v \; (f^*_{uv} - f_{uv}) + (f^*_{vu} - f_{vu}) = \; 0 \tag{9}$$
$$\forall u \neq s, t \; \sum_v f^*_{uv} - f_{uv} = 0 \tag{10}$$

which satisfies all conditions ((4),(5) and (6)) required by the flow in new graph.

Observation 2

Claim : The above flow $(f^* - f)$ can be written as a sum of *atmost* 2m flows, where each of them is a path from s to t.

Proof : This is simply because there can be a maximum of 2m different edges, and even if each represents a different flow we can only have 2m different flows from s to t.

Using the second observation I can say that, in every iteration I increase my flow by atleast $(|f^*|-|f|)/2m$. Hence

$$|f^{new}| \geq |f^{old}| + (|f^*| - |f|)/2m$$
$$|f^*| - |f^{new}| \leq (|f^*| - |f^{old}|)(1 - 1/2m)$$

So, in t iterations

$$|f^*|(1 - 1/2m)^t < 1$$
$$\text{as we began with } f^{old} = 0$$

This gives us,

$$t = 2m\ ln|f^*|$$

which is polynomial in size of input