CSL851: Algorithmic Graph Theory **Semester I 2013-14**

Lecture 10: August 26

Lecturer: Naveen Garg *Scribes: Abhinav Kumar*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

This lecture discusses the problem of adding as few edges as possible to an undirected graph to make it k-edge connected. This is also known as the Edge-Connectivity Augmentation problem. It also introduces the concept of Branchings and how to find the min-cost branching using Linear Programming.

10.1 Edge Connectivity Augmentation

Given a graph $G = (V,E)$, consider a partition of its vertices into disjoint sets $V_i s$. Let $\delta(V_i)$ be the number of edges going across V_i. Clearly this number should be atleast k for the graph to be k-edge connected. Clearly the following equation holds true :

$$\lceil\ (\ \sum_{i=1}^{p} (k - \delta(V_i))\ )/2\ \rceil \qquad \leq \qquad \textit{Minimum number of edges needed}$$

The division by 2 is because one edge is being counted twice i.e. one time each as going across from its source and destination sets respectively. We will show that the following equation is indeed an equality by doing a constructive proof (We will construct a partition for which this equality is satisfied) :

$$max(\ \lceil\ (\ \sum_{i=1}^{p} (k - \delta(V_i))\ )/2\ \rceil\ ) \qquad = \qquad \textit{Minimum number of edges needed}$$

Lemma 10.1 *Let E' be the set of minimum edges to be added to make the graph k-edge connected. Let $deg_{e'}(v)$ be the number of edges incident on v in E'. Knowing the $deg_{e'}(v)$ for each vertex v we can find the edge set E'.*

Proof: For the proof refer to scribe of Lecture 9 by MSK Swaroop. ■

Let $x(v)$ denote $deg_{e'}(v)$.
For a set of vertices U, let $x(U)$ denote $\sum_{v \epsilon U} x(v)$.

A set U is considered tight if the following relation holds :

$$x(U) = k - \delta(U)$$

Consider the following graph in which an extra vertex s has been added. From each vertex v, $x(v)$ edges have been drawn to the newly added vertex s. The *black* edges are the ones already in the graph and the *red* edges are those which have been drawn to the new vertex s.

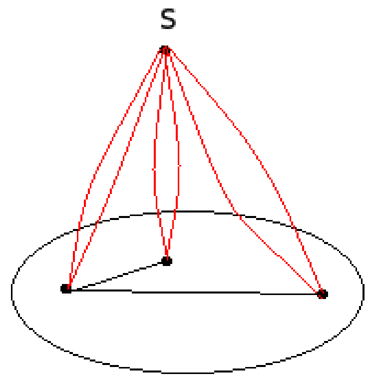

Figure 10.1:

Now consider the min-cut of this graph. Let the set not including s in the min-cut be U. Now pick a vertex in U whose $x(v) > 0$. Decrease the $x(v)$ value until either $x(v)$ becomes 0 or the set U becomes *tight*. If the $x(v)$ value had become 0 and the set U still hadn't become tight pick another vertex with $x(v) > 0$ and repeat the above step.

After this step either the set U would have become *tight* or it would have remained *non-tight*[In this case $x(v)$ values of all vertices in U would be 0].

Now pick a vertex with $x(v) > 0$ and which is not in U. Reduce $x(v)$ until v becomes part of a tight set or $x(v)$ becomes 0. Now pick another vertex with $x(v) > 0$ not in the tight sets formed until now and repeat this step on it.

At the end of the above procedure the following observation can be made :
Every v with $x(v) > 0$ is part of a tight set.

Lemma 10.2 *If A and B are tight, ($A \cup B$) and ($A \cap B$) are also tight.*

Proof:

1. $\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cup B)$ *[proved in earlier classes]*
2. $x(A) = k - \delta(A)$ *[A is tight]*
3. $x(B) = k - \delta(B)$ *[B is tight]*
4. $x(A \cup B) \geq k - \delta(A \cup B)$ *[We wouldn't have let the x value come down]*
5. $x(A \cap B) \geq k - \delta(A \cap B)$ *[We wouldn't have let the x value come down]*
6. $x(A) + x(B) = x(A \cup B) + x(A \cap B)$ *[x values are just numbers on vertices.]*

7. $2k - \delta(A) - \delta(B) \geq 2k - \delta(A \cup B) - \delta(A \cup B)$ *[Substituting values in 6 from 4 and 5]*

8. $\delta(A) + \delta(B) \leq \delta(A \cup B) + \delta(A \cup B)$

This implies that everything is an equality and both A∪B and A∩B are tight sets and *maximal tight sets are disjoint.*

■

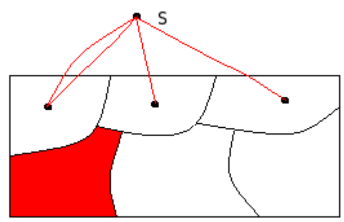

Figure 10.2:

In the above figure the red set has all vertices with $x(v)=0$ and may not necessarily be tight. However all other sets are tight.
We started with an arbitrary graph, added another vertex s to it and found the minimum number of edges which need to be added from each vertex to the vertex s to make the graph G∪s k-edge connected. If we apply Lovasz splitting-off lemma we will get a k-edge connected graph G. So how many edges did we have to add to make the graph G k-edge connected ? The answer is the following :

$$\lceil \ (\ \sum_{i=1}^{p} (k - \delta(V_i)) \ )/2 \ \rceil$$

This is because each of the white sets is *tight* and after applying Lovasz splitting-off lemma we get the factor of 2.
Note this is also the minimum number of edges we would need to make the graph k-edge connected if we look at the partition of disjoint tight sets we just got.
Hence the following equation holds true :

$$max(\lceil \ (\ \sum_{i=1}^{p} (k - \delta(V_i)) \ )/2 \ \rceil \ ) \quad = \quad \textit{Minimum number of edges needed}$$

10.2 Branchings

The *branching* problem is to find a minimal cost subset of edges in a graph such that there is a path from *r(root vertex)* to every vertex in V. The graph G being considered here is a directed graph.

Just as an example Prim's Algorithm for minimum spanning trees would fail in the following example:

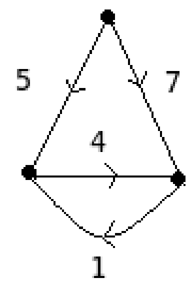

Figure 10.3:

Prim's would give answer 9 while correct answer is 8.

10.2.1 Algorithm

1. Take least incoming edge of every vertex except r.
2. We either get the tree or we get cycles.

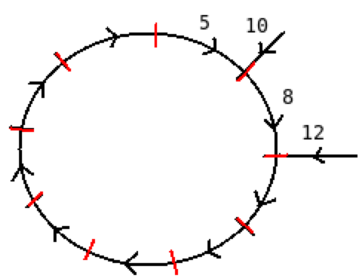

Figure 10.4:

3. Now we consider all the incoming edges to the cycle. We pick one of these edges and remove the other edge incoming to that vertex. For eg. In the figure above we could pick 12 and remove 8 (this would increase our cost by 12-8=4). We pick that edge the addition of which would result in the least rise in cost. Here the edge with weight 12 is preferred to the edge with weight 10.

10.2.2 Proof of correctness (Linear Programming)

The linear programming formulation for the branching problem is as follows :

10.2.2.1 Primal

1. x_e*=1 if e is in branching*
2. *min* $\sum c_e * x_e$

3. $\forall\ S \subseteq\ V\text{-}r,\ \sum_{e\in\delta_{in}(S)} x_e \geq 1$
4. $x_e \geq 0$

10.2.2.2 Dual

1. y_s *is the dual variable corresponding to the set s*
2. *max* $\sum y_s$
3. $\forall\ e,\ \sum_{s:e\in\delta_{in}(s)} y_s \leq c_e$

Following are the complementary-slackness conditions for the Linear Program :

1. x_e=1 only if constraint in the dual on the corresponding edge is tight.
2. Raise y_s only if x_e for an edge coming into s is 1.

Following are the steps for varying primal-dual variables $x_e,\ y_s$ keeping the slackness conditions satisfied :

1. For each vertex except r, set the x_e value of the least incoming edge to be 1. Simultaneously raise the y_s values of these singleton vertices to the value of the least incoming edge.
2. Now cycles may be formed by the edges with x_e=1. Raise the dual variable of the set of vertices in such a cycle by an amount = $\min_{edgesIntoCycle}(\ c_{edgeIntoCycle}$ - $c_{edgeRemoved}\ )$.
3. Set the x_e value of the edge just introduced into the branching to be 1.
4. In this way we are following the exact same procedure followed by the algorithm mentioned above. Also as we are keeping the slackness conditions satisfied we are bound to reach an optimal solution.