COMPSCI 632: Approximation Algorithms	October 23, 2017
Lecture 15	
Lecturer: Debmalya Panigrahi	*Scribe: Erin Taylor*

1 Overview

Last lecture we saw an algorithm for multi-way cut. We now consider the more general multi-cut. Given a graph and k terminal pairs $((s_1,t_1),....,(s_k,t_k))$ we want to find a minimum size cut disconnecting each pair. This problem is *NP*-hard and has no constant approximations. We consider how this problem relates to familiar flow problems.

We finally consider the maximum cut problem and introduce the concept of semi-definite programing.

2 Multi-Cut

We continue with our pipe system analogy. Consider edges to pipes, costs to be the cross-section area of the pipe, and our goal becomes minimizing the total volume of the system. Consider our LP formulation:

$$\begin{aligned} \min \sum_e c_e l_e & \\ \sum_{e \in P} l_e \geq 1 & \quad \forall i, \forall P(s_i,t_i) \\ l_e \geq 0 & \quad \forall e \in E \end{aligned}$$

We develop an algorithm, expanding on ideas from our algorithm for multi-way cut. Our algorithm works as follows: Take a radius, r_1, uniformly at random from $(0, \frac{1}{2})$, and take the cut associated with ball $B(s_1, r_1)$ around s_1. For the rest of the $i = 2,...,k$, if (s_i,t_i) are still connected, take $r_i \in (0, \frac{1}{2})$ uniformly at random and take the cut associated with $B(s_i, r_i)$. Continue until all pairs are cut.

Remark 1. *Notice that all balls we draw are disjoint. Additionally, a pair (s_i,t_i) can never be contained in a ball since each ball will have diameter < 1, and each pair are distance ≥ 1 away.*

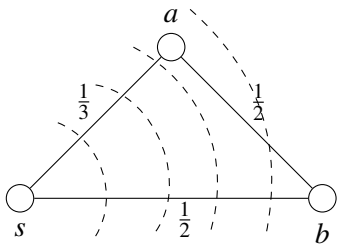
Lemma 1. *For one ball, the relationship between the volume and the size of the cut is:*

$$\frac{dV_r}{dr} \geq c(\delta(r))$$

where V_r is the volume of the ball with radius r and $c(\delta(r))$ is the cost of the cut.

Proof. For one ball, the volume being charged to that ball increases as the size of the cross-section (cost of the cut) increases. We interpret this as charging the cost of the cut to the volume of the ball.

Example 1. Let's consider an example to show why this $\geq$ is not equality. Consider a ball of different radii about s:



At $r = \frac{1}{3}$, $l_{(s,a)}$ adds $\frac{1}{3}$. However, r only increases by $\frac{1}{6}$ to include b, but at b, $l_{(s,b)}$ adds $\frac{1}{2}$, so here the volume increases faster. Notice this results from the fact that edges are not necessarily shortest paths. Physically, the pipes are not perpendicular to the direction the ball grows.

□

Theorem 2. *The algorithm gives an approximation factor of* $4\ln(k+1)$.

Proof. From our Lemma **??** we can say,

$$\frac{c(\delta(r))}{V_r} \leq \frac{\frac{dV_r}{dr}}{V_r}$$

Define $f(x) = \ln x$, so that $f(r) = \ln(V_r)$. Thus,

$$f'(r) = \frac{dV_r}{V_r} \geq \frac{c(\delta(r))}{V_r}$$

as $r \to 0$ to $1/2$:

$$\min_r \frac{c(\delta(r))}{V_r} \leq \min_r f'(r) \leq \frac{f(\frac{1}{2}) - f(0)}{\frac{1}{2} - 0} = 2[\ln(V_{1/2}) - \ln(V_0)] \leq 2\left[\ln\left(\frac{V^*}{0}\right)\right]$$

where V^* is the total volume of the pipe system.

However, notice that that unless we add some initial volume to the system, we cannot hope for a meaningful bound. We need to add enough to change the inclusion of V^* (so more than one unit at each s_i), but not so much that we lose a lot in our approximation (such as adding V^* to each s_i, where we would lose a factor of k). Thus, add $\frac{V^*}{k}$ to each s_i, and we will in total add V^* to the system, losing only a factor of 2.

Redefining V_r as $V_r = \text{vol}(B(s_i, r)) + \frac{V^*}{k}$ our previous inequality becomes:

$$\leq 2\left[\ln\left(\frac{V^*(1+\frac{1}{k})}{\frac{V^*}{k}}\right)\right] = 2\ln(k+1)$$

Thus, our total cost is $2\ln(k+1) * (\text{volume in balls}) = 2\ln(k+1)(2V^*)$, so our approximation factor is $4\ln(k+1)$.

□

3 Multi-Commodity Flow

Next, we consider flow problems and their relationship to Multi-Cut. The maximum commodity flow problem asks for a maximum flow given k different source and sink pairs: $((s_1,t_1),(s_2,t_2),....,(s_k,t_k))$. Suppose each edge has capacity c_e, then the following LP describes the problem:

$$\max \sum_{i=1}^{k} \sum_{p \in P(s_i,t_i)} f_p$$

$$\forall e \quad \sum_{i=1}^{k} \sum_{\substack{p \in P(s_i,t_i) \\ e \in P}} f_p \leq c_e$$

The dual of this problem is exactly the fractional multi-cut problem. The duality gap is 1 only if the instance is a single commodity flow, otherwise the gap is $O(\ln(k))$.

- Consider single commodity flow, with unit capacities, there is no gap between the integral and fractional flows, the proof of this is just the Ford-Fulkerson algorithm.
- Consider when $k = 2$, with unit capacities. The maximum flow value is either 1 or 2. Define paths $p_1 \in P(s_1,t_1)$ and $p_2 \in P(s_2,t_2)$, if there exist edge disjoint p_1 and p_2 then flow is 2, otherwise it is 1. This is the edge disjoint path problem, known to be NP-hard.
- One possible idea to solve multi-commodity flows is to create a supersource and a supersink and just find the maximum flow between them. However, this allows flows to be sent from (s_i,t_j) where $i \neq j$.

If we care about fairness, i.e. we want to prevent starving, instead of maximizing a sum we could maximize the minimum. This LP:

$$\max \ \lambda$$

$$\forall e \quad \sum_{i=1}^{k} \sum_{\substack{P \in (s_i,t_i) \\ e \in P}} f_p \leq c_e$$

$$\sum_{P \in P(s_i,t_i)} \geq \lambda$$

Define dual variables l_e for the first constraint and y_i for the second constraint. Then we can write the dual LP:

$$\min \sum_e c_e l_e$$

$$\forall P \in P(s_i,t_i) \sum_{e \in P} l_e - y_i \geq 0$$

$$\sum_{i=1}^{k} y_i \geq 1$$

If we consider y_i to be the distance from (s_i,t_i) under function l_e, then we can rewrite the previous two constraints as

$$\sum_{1}^{k} d_l(s_i,t_i) \geq 1$$

We can interpret this LP as wanting to give lengths to edges such that we minimize volume, but ensuring that the sum of distances must be 1. The integral version of this would be to cut any pair.

4 Max-Cut and Semi-definite Programming

In this section, we consider a different cut problem, maximum cut: given a graph find a partition of the vertices into two sets such that the size of the cut is maximized. If edges have weights, maximize the weight of the cut. It is easy to get a $1/2$-approximation to this problem.

Start with any cut. Switch a vertex to the other set if it increases the size of the cut, this is a local search algorithm. Notice the algorithm must terminate because you only make a move if it increases your objective. At the end of the algorithm, for each vertex, its degree across the cut is greater than its degree inside its set, otherwise you would have moved it. Thus, at least $E/2$ edges must be part of the cut. This is a $1/2$-approximation with respect to the number of edges, E. Randomly assigning a vertex to a side also gives a $1/2$-approximation.

These approximations are compared against the total number of edges, but in actuality the maximum cut might be much smaller. We will see that a semi-definite program gives a much better lower bound.

Definition 1. *We can write a semi-definite program as follows:*

$$\begin{aligned} \min\ & c^T x \\ & Ax \geq b \\ & [x] \succeq 0 \end{aligned}$$

where $[x]$ is the matrix defined by vector variables. We can rewrite the above:

$$\begin{aligned} \min & \sum_{i,j} c_{ij}(v_i \cdot v_j) \\ & \sum_k a_{ijk}(v_i \cdot v_j) \geq b_k \\ & v_i \cdot v_i = 1 \quad \forall i \end{aligned}$$

If each vertex is a unit vector variables, we are really embedding the vertices on a unit hypersphere.

Remark 2. *A is positive semi-definite if $\forall y : y^T Ay \geq 0$. This is equivalent to all eigenvalues $\lambda_i \geq 0$, and $A = V^T V$ for matrix $V_{k\times n}$ where $k \leq n$. Thus, each entry is really an inner product of two vectors.*

Consider max-flow as assigned each vertex to the one dimensional vector -1 or 1, i.e. $v_i \cdot v_i = 1$. Then we can formulate the problem as follows:

$$\begin{aligned} \max\ & \frac{1}{2} \sum_{(i,j)\in E} c_{ij}(1 - v_i \cdot v_j) \\ & x_i \in \{-1, 1\} \end{aligned}$$

Next class, we will see how to relax this to a semi-definite program, and how to round using a random splitting hyperplane.

5 Summary

In this lecture we saw an algorithm for multi-cut. We drew disjoint balls, removing crossing edges until all pairs were disconnected. By charging the costs of edges cut to the volume in the balls, this becomes the total volume in the system. We got a $O(\ln(k))$ approximation by cutting with the minimum radius. We then explored the relationship between this problem and flows, and their varying difficulty. Finally, we used the max-cut problem to motivate our introduction of semi-definite programming.