18.1 Introduction

Today we’re going to talk about a cut problem known as Multicut which is even more general than Multiway Cut. We very briefly discussed this at the end of last lecture, where I described how to use techniques that we already know to design an $O(\log n)$-approximation. While I did this very quickly, all the details are in the lecture notes from last time. Today we’re going to improve this slightly to give an $O(\log k)$-approximation.

18.2 Definition and Relaxation

Definition 18.2.1 In the Multicut problem, we are given a graph $G = (V, E)$ with costs $c : E \to \mathbb{R}^+$, and $k$ pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of nodes. A feasible solution is a set $F \subseteq E$ such that $s_i$ and $t_i$ are not connected in $G \setminus F$ for all $i \in [k]$. The objective is to minimize $\sum_{e \in F} c(e)$.

For the remainder of the day, we’re going to prove the following theorem:

Theorem 18.2.2 There is an $O(\log k)$-approximation algorithm for Multicut.

We will use $\mathcal{P}_i$ to denote the set of all $s_i$-$t_i$ paths. The problem admits the following LP relaxation:

\[
\text{minimize: } \sum_{e \in E} c(e) \cdot x_e \quad \text{(MULTICUT-LP)}
\]

subject to:

\[
\sum_{e \in P} x_e \geq 1 \quad \forall i \in [k], \forall P \in \mathcal{P}_i \quad (18.2.1)
\]

\[
0 \leq x_e \leq 1 \quad \text{for each edge } e \in E \quad (18.2.2)
\]

Note: As with multiway cut, we can solve this LP in polytime via ellipsoid, using shortest path (for each $\mathcal{P}_i$) to separate. For the remainder, we will use $x$ to refer to the solution of the LP, and set $V^* = \sum_{e \in E} c(e)x_e$ as the value of the solution.

Definition 18.2.3 Let $d : V \times V \to \mathbb{R}^+$ be the shortest path metric using the LP solution $\bar{x}$ for the edge lengths.

Definition 18.2.4 For all $S \subseteq V$, let $\delta(S) = E(S, \bar{S})$ denote all edges that have exactly one endpoint in $S$.

Definition 18.2.5 For all sets of edges $E' \subseteq E$, let $c(E') = \sum_{e \in E'} c(e)$. 

18.3 Rounding

To move forward, we’re going to take inspiration from a physical metaphor. This of each edge as a “pipe”. We’re thinking of \( x_e \) as the length of \( e \), so if we think of \( c(e) \) as the “cross-sectional area”, then the “volume” of an edge would be \( c(e)x_e \). This motivates the following definition.

**Definition 18.3.1** (Volume)

\[
V(s_i, r) = \frac{V^*}{k} + \sum_{\{u,v\} \in E, \quad u,v \in B(s_i, r)} c(e)x_e + \sum_{\{u,v\} \in E, \quad u \in B(s_i, r), \quad v \notin B(s_i, r)} c(e)(r - d(s_i, u))
\]

The second term above should be thought of as the volume of all edge-pipes fully inside the ball around \( s_i \), and the third as (a lower bound for) the volume contained in \( B_G(s_i, r) \) of edge-pipes leaving the ball. The first term is included to make later calculations easier.

The next lemma is the main technical piece.

**Lemma 18.3.2** (Region-Growing Lemma) For all \( i \in [k] \), we can find in polytime a value \( 0 \leq r < \frac{1}{2} \) such that:

\[
c(\delta(B(s_i, r))) \leq 2 \ln(k + 1) \cdot V(s_i, r)
\]

Before we prove this lemma, let’s show how to approximate MULTICUT if we assume that it is true.

**Algorithm 1** Constructing an integer solution

```plaintext
Init: \( F = \emptyset \)
for \( i = 1 \) to \( k \) do
    if \( s_i, t_i \) connected in \( G \) then
        Let \( r_i \in [0, \frac{1}{2}] \) be the \( r \) value from the region-growing lemma.
        \( F \leftarrow F \cup \delta(B(s_i, r_i)) \)
        Remove \( B(s_i, r_i) \) and all incident edges from the graph.
    end if
end for
return \( F \)
```

One important note to clarify this, since we’re changing the graph throughout this algorithm: distances, balls and volumes are with respect to the *current* graph, not the original.

**Theorem 18.3.3** The output \( F \) from Algorithm 1 is feasible.

**Proof:** The only way this might not be feasible is if some \( s_i - t_i \) pair are both in \( B(s_j, r) \) for some \( j \). But this cannot happen since \( r < 1/2 \) and \( d(s_i, t_i) \geq 1 \) throughout the algorithm.

**Theorem 18.3.4** \( c(F) \leq 4 \ln(k + 1)V^* \leq 4 \ln(k + 1) \cdot OPT \).

**Proof:** Let’s do some definitions.

- Let \( B_i \) be \( B(s_i, r_i) \) in iteration \( i \) (if we did not create such a ball in iteration \( i \) because \( s_i \) and \( t_i \) were already separated, let \( B_i = \emptyset \)). Note that since the algorithm changes the
graph throughout the algorithm, this might not have been $B(s_i, r_i)$ at the beginning of the algorithm.

- Similarly, let $F_i = \delta(B(s_i, r_i))$ be the edges removed by the algorithm in iteration $i$. Then clearly $F = \cup_{i=1}^k F_i$, and $F_i \cap F_j = \emptyset$ for all $i \neq j$.

- Let $V_i = \sum_{e=\{u,v\}:u,v \in B_i} c(e)x_e + \sum_{e \in E} c(e)x_e$ be the total volume of edges removed in iteration $i$. Note that $V_i \geq V(s_i, r_i) - \frac{V^*}{k}$, since $V_i$ contains the full volume of edges in $F_i$ while $V(s_i, r_i)$ contains only part of their volume (but with an additional $V^*/k$).

Moreover, every edge contributes to $V_i$ for at most one value of $i$, since the first time at least one of the endpoints is in $B_i$, the edge is removed from the graph. Thus $\sum_{i=1}^k V_i \leq V^*$

Now note that every edge in $F$ is in exactly one $F_i$ by our definition of the $F_i$’s, and moreover the value $r_i$ was chosen from the region growing lemma. Thus we get that

$$c(F) = \sum_{i=1}^k c(F_i) \leq (2\ln(k+1))\sum_{i=1}^k V(s_i, r_i)$$

$$\leq (2\ln(k+1))\sum_{i=1}^k \left(V_i + \frac{V^*}{k}\right)$$

$$\leq 4\ln(k+1) \cdot V^*,$$

as claimed.

So now it only remains to prove the Region Growing Lemma (Lemma 18.3.2). For the rest of today, let $c(r) = c(\delta(B(s_i, r)))$ and let $V(r) = V(s_i, r)$.

**Proof of Lemma 18.3.2.** We’re eventually going to get a deterministic algorithm, but let’s start with a randomized algorithm: choose $r$ uniformly at random from $[0, 1/2]$. We want to show that if we do this, then $E\left[\frac{c(r)}{V(r)}\right] \leq 2\ln(k+1)$.

Order $B(s_i, \frac{1}{2})$ as $\{v_1, \ldots, v_m\}$, where $r_j = d(s_i, v_j)$, and $0 \leq r_1 \leq r_2 \leq \cdots \leq r_m < \frac{1}{2}$. We also define $r_0 = 0$ for later calculations.

Surprisingly, we’re going to do a bunch of calculus to prove this. I’m going to abuse calculus a bit here – see the book for the more formally correct way of doing this. Consider the function $V(r)$, which (just to recall) is

$$V(r) = \frac{V^*}{k} + \sum_{e=\{u,v\} \in E \atop u,v \in B(s_i, r)} c(e)x_e + \sum_{e=\{u,v\} \in E \atop u \in B(s_i, r) \atop v \notin B(s_i, r)} c(e)(r - d(s_i, u)).$$

Unfortunately, $V(r)$ is not continuous or differentiable, since there can be discontinuities at the values $\{r_j\}$. But let’s pretend like it’s differentiable. Note that for $r \in (r_j, r_{j+1})$, for any $j$, it is in fact differentiable with derivative $\frac{d}{dr}V(r) = c(r)$. This is because the first and second terms are constant in this range of $r$, so we just need to care about the third term, which gives exactly $c(r)$.
Now we can use calculus to figure out the “average” value of \( \frac{c(r)}{V(r)} \) over \([0, \frac{1}{2}]\):

\[
\frac{1}{2} \int_0^{1/2} \frac{c(r)}{V(r)} \, dr = 2 \int_0^{1/2} \frac{1}{V(r)} \cdot \frac{dV(r)}{dr} \, dr
\]

\[
= 2 \int_0^{1/2} \frac{1}{V(r)} \, dV(r)
\]

\[
= 2(\ln(V(1/2)) - \ln(V(0)))
\]

\[
= 2 \ln \left( \frac{V(1/2)}{V(0)} \right)
\]

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

It then would follow from the mean value theorem that there exists some \( r \in [0, \frac{1}{2}] \) achieving the average value. For this \( r \) we would then have

\[ c(r) \leq 2 \ln(k + 1) \]

\[ V(r) \]

Before, we concluded by saying that the MVT allowed us to find an \( r \) achieving the average value. Here, because \( V(r) \) is increasing and \( c(r) \) is constant over each \([r_j, r_{j+1}]\) interval, we can say that the smallest value of \( \frac{c(r)}{V(r)} \) will occur at some \( r_j^- \). By the above, for \( r = r_j^- \) we will then have that

\[ c(r) \leq 2 \ln(k + 1) V(r), \text{ as desired.} \]

And note that there are only \( m \leq n \) different values of \( r_j \), so we can just check each one and deterministically find the best.