CMPUT 675: Approximation Algorithms **Fall 2014**

Lecture 26 (Nov 7 & 12): k-MST and Prize Collecting Steiner Tree

Lecturer: Zachary Friggstad *Scribe: Zachary Friggstad*

26.1 The Problems

We discussed two problems in these lectures. We first recall their definitions, present some related linear programs, and collect some basic facts before delving into the algorithms.

Definition 1 *In the k-MST problem, we are given a metric graph $G = (V \cup \{r\}, E)$ with distances $d(e), e \in E$ and an integer $k \leq |V|$. The goal is to find the cheapest $F \subseteq E$ such that at least k nodes in V are in the same component as r in the graph $(V \cup \{r\}, F)$.*

Clearly a minimum-cost solution is a tree.

Definition 2 *In the* Price Collecting Steiner Tree *problem, we are given a metric graph $G = (V \cup \{r\}, E)$ with distances $d(e), e \in E$ and penalties $\pi_v \geq 0, v \in V$. The goal is to find a set of edges F to buy and a set of nodes $P \subseteq V$ to discard of minimum total value $\sum_{e \in F} d(e) + \sum_{v \in P} \pi_v$ such that every vertex not in P is connected to r in $(V \cup \{r\}, F)$.*

Throughout, we will let $n = |V|$ so there are $n+1$ nodes in total in $G = (V \cup \{r\}, E)$. We will also use $d(F)$ to denote $\sum_{e \in F} d(e)$ and $\pi(P)$ to denote $\sum_{v \in P} \pi_v$. As with the Steiner Tree problem, we can reduce the general non-metric case to the metric case by working in the shortest path metric, so assuming G is a metric does not lose any generality.

26.1.1 Linear Programs

A natural LP relaxation for the k-MST problem is the following. Here, $x_e = 1$ corresponds to us buying edge e and $z_v = 1$ corresponds to us choosing to not connect $v \in V$ to r.

$$
\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} d(e) \cdot x_e \\
\text{subject to:} \quad & \sum_{e \in \delta(S)} x_e + z_v \geq 1 && \text{for each } v \in V \text{ and each } \{v\} \subseteq S \subseteq V - \{r\} \\
& \sum_{v \in V} z_v \leq n - k \\
& \mathbf{x}, \mathbf{z} \geq 0
\end{aligned}
\qquad \textbf{(LP-kMST)}
$$

Unfortunately, (**LP-kMST**) has a bad integrality gap. For example, suppose G has an edge (r, v_1) with cost n and $n-1$ edges $(v_1, v_i), 2 \leq i \leq n$ each with cost 1 then the optimum integer solution for $k = 2$ is $n+1$ but we can set $z_{v_i} = (n-2)/n$ and $x_e = 2/n$ for each edge e which is a feasible LP solution with cost ≤ 2.

We will still use (**LP-kMST**) to design a constant-factor approximation. To do this, we "Lagrangify" the cardinality constraint. In particular, for any $\lambda \geq 0$ we consider the following LP where we call λ the Lagrangean multiplier.

$$\begin{array}{lrcll} \text{minimize :} & \sum_{e \in E} d(e) \cdot x_e + \lambda \cdot \left(\sum_{v \in V} z_v - (n-k) \right) & & & \\ \text{subject to :} & \sum_{e \in \delta(S)} x_e + z_v & \geq & 1 & \text{for each } v \in V \text{ and each } \{v\} \subseteq S \subseteq V - \{r\} \\ & \mathbf{x}, \mathbf{z} & \geq & 0 & \end{array}$$

$(\textbf{LP-kMST}(\boldsymbol{\lambda}))$

Finally, we consider the LP that is obtained by dropping the constant term from the objective function of $(\textbf{LP-kMST}(\boldsymbol{\lambda}))$.

$$\begin{array}{lrcll} \text{minimize :} & \sum_{e \in E} d(e) \cdot x_e + \sum_{v \in V} \lambda \cdot z_v & & & \\ \text{subject to :} & \sum_{e \in \delta(S)} x_e + z_v & \geq & 1 & \text{for each } v \in V \text{ and each } \{v\} \subseteq S \subseteq V - \{r\} \\ & \mathbf{x}, \mathbf{z} & \geq & 0 & \end{array}$$

$(\textbf{LP-PCST}(\boldsymbol{\lambda}))$

Note that $(\textbf{LP-PCST}(\boldsymbol{\lambda}))$ is an LP relaxation of the PRIZE COLLECTING STEINER TREE instance defined over G where every $v \in V$ has penalty $\pi_v = \lambda$.

Notation

- OPT - The optimum solution value to the original k-MST instance.
- $OPT_k(\lambda)$ - The optimum solution value of $(\textbf{LP-kMST}(\boldsymbol{\lambda}))$.
- $OPT_{ST}(\lambda)$ - The optimum solution value of $(\textbf{LP-PCST}(\boldsymbol{\lambda}))$.

26.1.2 Observations

Lemma 1 *For any $\lambda \geq 0$, $OPT_k(\lambda) \leq OPT$.*

Proof. The natural $\{0,1\}$ solution corresponding to the optimum k-MST solution is feasible for $(\textbf{LP-kMST}(\boldsymbol{\lambda}))$. The second term of the objective function is nonpositive because at most $n-k$ nodes are not connected to the root r and $\lambda \geq 0$. ■

Lemma 2 *For any $\lambda \geq 0$, $OPT_k(\lambda) + \lambda \cdot (n-k) = OPT_{ST}(\lambda)$.*

Proof. Their feasible solutions are the same and their objective functions differ by $\lambda \cdot (n-k)$. ■

The following will be used in Section 26.2 and its proof appears in Section 26.3.

Theorem 1 *For any $\lambda \geq 0$, we can find a set of edges F with corresponding nodes P not connected to r in $(V \cup \{r\}, F)$ such that $d(F) + 2 \cdot \lambda \cdot |P| \leq 2 \cdot OPT_{ST}(\lambda)$.*

Such an algorithm is called a "Lagrangean multiplier preserving approximation". This is stronger than saying we have an LP-based 2-approximation for the PRIZE COLLECTING STEINER TREE instance on G where all penalties are λ. The factor 2 in front of the penalties in the approximation guarantee in Theorem 1 is crucial in the upcoming k-MST approximation.

26.2 k-MST

We present a $(5+\epsilon)$-approximation with running time that is polynomial in the size of G and $\log \frac{1}{\epsilon}$. A short discussion follows on how to get a 5-approximation (without the ϵ).

As a pre-processing step, we guess the furthest node that is spanned by the optimum solution and discard all farther nodes. By this, we mean we try all nodes v as our guess, run the following algorithm, and keep the best solution found over all guesses.

So, we assume that all $v \in V$ satisfy $d(r,v) \le OPT$ from now on. It is easy to check if there is a 0-cost solution (i.e. if at least k nodes have distance 0 to r), so we also assume $OPT > 0$. Let $\delta \le OPT$ be the minimum non-zero distance $d(r,v)$ over $v \in V$.

For any $\lambda \ge 0$, consider what happens when we invoke the approximation algorithm stated in Theorem 1. When $\lambda = 0$, then $F = \emptyset$ and $P = V$ is returned and when λ is very large, say $2 \cdot \mathrm{MST}(G)+1$, then a spanning tree F of $V \cup \{r\}$ and $P = \emptyset$ is returned. This is because any solution which discards even a single node when the penalty is Δ is more than twice as expensive as the minimum spanning tree of G, so it cannot be a 2-approximate solution. This justifies the binary search in Algorithm 1 below.

Algorithm 1 k-MST Approximation (assuming $0 < d(r,v) \le OPT$ for each $v \in V$)

Binary search for $\lambda_1, \lambda_2 \in [0, 2 \cdot MST(G)+1]$ such that:
  a) $\lambda_1 \le \lambda_2 \le \lambda_1 + \frac{\epsilon \cdot \delta}{4 \cdot n}$
  b) F_i, P_i are returned when using the approximation in Theorem 1 invoked with uniform penalties λ_i
  c) $|P_1| \ge n-k \ge |P_2|$
if either $|P_1|$ or $|P_2|$ equals $n-k$ **then**
  return the respective F_1 or F_2
end if
$\alpha_1 \leftarrow \frac{(n-k)-|P_2|}{|P_1|-|P_2|}$
$\alpha_2 \leftarrow \frac{|P_1|-(n-k)}{|P_1|-|P_2|}$
if $\alpha_2 \ge 1/2$ **then**
  return F_2 (**Note**: F_2 is feasible because $|P_2| \le n-k$)
else
  double and shortcut F_2 to get a cycle C spanning the nodes included in F_2 but not F_1
    (**Note**: C has at least $|P_1| - |P_2|$ nodes)
  find the cheapest subpath P of length $|P_1| - (n-k)$ in C
  return $F_1 \cup P \cup \{(r,v)\}$ where v is some node on P (**Note**: this solution has exactly $n-k$ nodes)
end if

The number of iterations of the binary search routine is polynomial in the size of G and $\log \frac{1}{\epsilon}$. Clearly the rest of the algorithm runs in polynomial time.

26.2.1 Analysis

We start with the easy case where either F_1 or F_2 excludes exactly $n-k$ nodes (the first **return** statement).

Lemma 3 *If $|P_1|$ or $|P_2|$ has size exactly $n-k$, then the cost of the respective F_1 or F_2 is at most $2 \cdot OPT$.*

Proof. Suppose $|P_1| = n-k$, essentially the same proof will work when $|P_2| = n-k$. By the Lagrangean preserving property guaranteed in Theorem 1, we know $d(F_1) + 2 \cdot \lambda_1 \cdot |P_1| \le 2 \cdot OPT_{ST}(\lambda_1)$. Rearranging, we have

$$\begin{aligned} d(F_1) &\leq 2 \cdot (OPT_{ST}(\lambda_1) - \lambda_1 \cdot (n-k)) \\ &= 2 \cdot OPT_k(\lambda_1) \quad \text{(by Lemma 2)} \\ &\leq 2 \cdot OPT \qquad\quad \text{(by Lemma 1)} \end{aligned}$$

■

Now suppose $|P_1| > n-k > |P_2|$. The following observation can be verified in a straightforward manner.

Observation 1 *The coefficients α_1, α_2 satisfy the following.*

- $\alpha_1, \alpha_2 \geq 0$
- $\alpha_1 + \alpha_2 = 1$
- $\alpha_1 \cdot |P_1| + \alpha_2 \cdot |P_2| = n-k.$

These observations say that in some sense the two solutions F_1, F_2 are "feasible on average". Of course, this is not a precise statement but the intuition works well: the following lemma states that "on average" their cost is bounded.

Lemma 4 $\alpha_1 \cdot d(F_1) + \alpha_2 \cdot d(F_2) \leq \left(2 + \frac{\epsilon}{2}\right) \cdot OPT$

Proof. We start by noting two points:

1. $OPT_{ST}(\lambda_1) \leq OPT_{ST}(\lambda_2)$
   Indeed, the optimal solution to (**LP-PCST(λ)**) for $\lambda = \lambda_2$ is feasible for the same LP with $\lambda = \lambda_1$ and is no more expensive because $\lambda_1 \leq \lambda_2$.
2. $(\lambda_2 - \lambda_1) \cdot \alpha_2 \cdot |P_2| \leq \frac{\epsilon}{4} \cdot OPT$
   From the binary search, we have
   $$\lambda_2 - \lambda_1 \leq \frac{\epsilon \cdot \delta}{4 \cdot n} \leq \frac{\epsilon \cdot OPT}{4 \cdot n}.$$
   The latter bound is because δ is the minimum nonzero distance in the metric. Since $\alpha_2 \leq 1$ and $|P_2| \leq n$, then the claimed bound holds.

Therefore

$$\begin{aligned}
& \alpha_1 \cdot d(F_1) + \alpha_2 \cdot d(F_2) \\
\leq\; & 2 \cdot (\alpha_1 \cdot OPT_{ST}(\lambda_1) - \lambda_1 \cdot \alpha_1 \cdot |P_1| + \alpha_2 \cdot OPT_{ST}(\lambda_1) - \lambda_2 \cdot \alpha_2 \cdot |P_2|) && \text{(by Theorem 1)} \\
\leq\; & 2 \cdot ((\alpha_1 + \alpha_2) \cdot OPT_{ST}(\lambda_2) - \lambda_1 \cdot \alpha_1 \cdot |P_1| - \lambda_2 \cdot \alpha_2 \cdot |P_2|) && \text{(Point 1 above)} \\
=\; & 2 \cdot (OPT_{ST}(\lambda_2) - \lambda_2 \cdot (\alpha_1 \cdot |P_1| + \alpha_2 \cdot |P_2|) + (\lambda_2 - \lambda_1) \cdot \alpha_1 \cdot |P_2|) && \text{(rearranging and recalling } \alpha_1 + \alpha_2 = 1) \\
=\; & 2 \cdot (OPT_{ST}(\lambda_2) - \lambda_2 \cdot (n-k) + (\lambda_2 - \lambda_1) \cdot \alpha_1 \cdot |P_2|) && \text{(recalling } \alpha_1 \cdot |P_1| + \alpha_2 \cdot |P_2| = n-k) \\
\leq\; & 2 \cdot OPT + 2 \cdot (\lambda_2 - \lambda_1) \cdot \alpha_1 \cdot |P_2| && \text{(Lemmas 1 and 2)} \\
\leq\; & \left(2 + \tfrac{\epsilon}{2}\right) \cdot OPT && \text{(Point 2 above)}
\end{aligned}$$

■

The second easiest case is when $\alpha_2 \geq 1/2$. Algorithm 1 returns a solution that excludes $|P_2| \leq n-k$ nodes from the tree F_2 (*i.e.* it is a feasible solution). We claim that it is also a relatively cheap solution.

Lemma 5 *If $\alpha_2 \geq 1/2$, then $d(F_2) \leq (4+\epsilon) \cdot OPT$.*

Proof.

$$d(F_2) \leq 2 \cdot \alpha_2 \cdot d(F_2) \leq 2 \cdot (\alpha_1 \cdot d(F_1) + \alpha_2 \cdot d(F_2)) \leq (4+\epsilon) \cdot OPT.$$

The last bound is by Lemma 4. ■

We are left with the "tricky" case when $\alpha_2 < 1/2$. As with Lemma 5, we can show that F_1 has low cost. However, it is not a feasible solution which is why Algorithm 1 *grafts* on the path P to turn it into a feasible solution. We first argue that this procedure is valid (i.e. the C contains enough nodes) and then bound the cost of the returned solution.

First, we claim that F_2 and, thus, C spans at least $|P_1| - |P_2|$ nodes that are not spanned by F_1. This is simple counting; if A_1, A_2 denote the nodes spanned by F_1, F_2 respectively, then the number of nodes that are spanned by F_2 but not by F_1 is bounded as follows:

$$|A_2 - A_1| \geq |A_2| - |A_1| = (n - |P_2|) - (n - |P_1|) = |P_1| - |P_2|.$$

Also recall that $|P_2| \leq n - k$ so it makes sense to ask for the cheapest subpath of C that spans $|P_1| - (n-k) \leq |P_1| - |P_2|$ nodes.

Having justified the execution of the algorithm, we bound the final cost.

Lemma 6 *If $\alpha_1 > 1/2$, then the cost of the solution returned by the algorithm is at most $(5+\epsilon) \cdot OPT$.*

Proof. By our preprocessing step (where we discarded nodes u with $d(r,u) > OPT$), we have $d(r,v) \leq OPT$ where v is the node indicated in the last step. It suffices to bound $d(F_1) + d(P)$ by $(4+\epsilon) \cdot OPT$.

Since we obtain C by doubling and shortcutting, then $d(C) \leq 2 \cdot d(F_2)$. Since P is the cheapest subpath of C with length $|P_1| - (n-k)$ and since C has $|P_1| - |P_2|$ nodes, then

$$d(P) \leq \frac{|P_1| - (n-k)}{|P_1| - |P_2|} \cdot d(C) = \alpha_2 \cdot d(C) \leq 2 \cdot \alpha_2 \cdot d(F_2).$$

Because $\alpha_1 \geq 1/2$, we have:

$$d(F_1) + d(P) \leq 2 \cdot (\alpha_1 \cdot d(F_1) + \alpha_2 \cdot d(F_2)) \leq (4+\epsilon) \cdot OPT.$$

Again, the last bound is by Lemma 4. ■

26.2.2 Removing the ϵ

Theorem 1 can be improved to the bound $d(F) + (2 - 1/n) \cdot \lambda \cdot |P| \leq (2 - 1/n) \cdot OPT_{ST}(\lambda)$. Comments on this will be made at the end of the PRIZE COLLECTING STEINER TREE approximation below.

Run the binary search to find λ_1, λ_2 as before, except with the bound

$$\lambda_2 - \lambda_1 \leq \frac{\delta}{2 \cdot n^2}.$$

Lemma 4 then guarantees

$$\alpha_1 \cdot d(F_1) + \alpha_2 \cdot d(F_2) \leq (2 - 1/n) \cdot OPT + (2 - 1/n) \cdot (\lambda_2 - \lambda_1) \cdot \alpha_2 \cdot |P_2| \leq 2 \cdot OPT.$$

Using this cleaner bound in the rest of the proof yields the 5-approximation.

26.3 Prize Collecting Steiner Tree

We begin by considering an LP relaxation for PRIZE COLLECTING STEINER TREE that is not the obvious generalization of (**LP-PCST(λ)**) to arbitrary penalties π. Here, for every subset $S \subseteq V - \{r\}$ we have a variable z_S. Recall that $\pi(S)$ denotes $\sum_{v \in S} \pi_v$.

$$\begin{array}{lrcll} \text{minimize :} & \displaystyle\sum_{e \in E} c_e \cdot x_e + \sum_{\emptyset \subsetneq S \subseteq V} \pi(S) \cdot z_S & & & \\ \text{subject to :} & \displaystyle\sum_{e \in \delta(S)} x_e + \sum_{S \subseteq R \subseteq V - \{r\}} z_R & \geq & 1 & \text{for each } \emptyset \subsetneq S \subseteq V \\ & x, z & \geq & 0 & \end{array} \qquad \textbf{(LP-PCST)}$$

This relaxation contains exponentially many constraints and variables. However, the algorithm we will consider is a primal-dual algorithm which still runs in polynomial time. We let OPT_{PCST} denote the optimum solution value of (**LP-PCST**).

The dual of (**LP-PCST**) is the following.

$$\begin{array}{lrcll} \text{maximize :} & \displaystyle\sum_{\emptyset \subsetneq S \subseteq V} y_S & & & \\ \text{subject to :} & \displaystyle\sum_{\substack{\emptyset \subsetneq S \subseteq V \\ \text{s.t. } e \in \delta(S)}} y_S & \leq & c_e & \text{for each } e \in E \\ & \displaystyle\sum_{\emptyset \subsetneq R \subseteq S} y_R & \leq & \pi(S) & \text{for each } \emptyset \subsetneq S \subseteq V \\ & y & \geq & 0 & \end{array} \qquad \textbf{(Dual-PCST)}$$

While the k-MST results required approximation algorithms relative to (**LP-PCST(λ)**), it suffices to use (**LP-PCST**).

Lemma 7 *Suppose $\lambda \geq 0$ is such that $\pi_v = \lambda$ for each $v \in V$. Then $OPT_{ST}(\lambda) = OPT_{PCST}$.*

Proof. Given an optimum solution (x^*, z^*) to (**LP-PCST**), we construct a feasible solution to (**LP-PCST(λ)**) by using the same x^* and setting $z_v = \sum_{S: v \in S} z_S$. It is easy to verify this is feasible for (**LP-PCST(λ)**) with the same cost, so $OPT_{ST}(\lambda) \leq OPT_{PCST}$.

Conversely, suppose (x^*, z^*) is an optimum solution to (**LP-PCST(λ)**). Order V so that $z_{v_1} \leq z_{v_2} \leq \ldots \leq z_{v_n}$. For each $1 \leq i \leq n$ let $S_i = \{i, \ldots, n\}$ and set $z_{S_i} = z_{v_i} - z_{v_{i-1}}$ (if $i = 1$ then just set $z_{S_1} = z_{v_1}$). It is easy to verify this yields a feasible solution to (**LP-PCST**) with the same cost, so $OPT_{PCST} \leq OPT_{ST}(\lambda)$. ■

26.3.1 The Algorithm

Now we focus on the main algorithm, which works for general penalties π.

Theorem 2 *There is a polynomial-time algorithm that finds a feasible* PRIZE COLLECTING STEINER TREE *solution (F, P) such that $c(F) + 2 \cdot \pi(P) \leq 2 \cdot OPT_{PCST}$.*

The algorithm is a primal-dual algorithm. We will grow moats around the nodes in V, much like in the case of STEINER FOREST, and grow them until they connect to r. However, each moat will come with an associated *charge* that depletes over time. If the charge is completely depleted, then we give up on connecting that moat to r. Once this phase is done, we prune the solution to discard edges not connected to r and edges whose deletion produces a component that had 0 charge at some point in the algorithm.

In Algorithm 2, we grow a forest F alongside a feasible dual y. An active component will be a connected component C of F such that the dual constraint for C is slack. Note that if C contains r, then it is not an active component. Call this component the *root component* of F.

Each component C will be associated with a charge $\gamma(C)$ that depletes over time. These charges keep track of the slack of the dual constraint for a component C of F; in particular, C is active if and only if $\gamma(C) > 0$.

Algorithm 2 Lagrangean multipler preserving 2-approximation for PRIZE COLLECTING STEINER TREE

$F \leftarrow \emptyset$
$\gamma(\{v\}) = \pi_v$ for each $v \in V$
$\mathcal{C} = \{\{v\} : v \in V\}$
$y \leftarrow 0$
while there are still active components **do**
  Simultaneously raise y_C and decrease $\gamma(C)$ for each active component C (at the same rate for each) until:
  1) $\gamma(C)$ becomes 0 for some active component C,
    Do nothing (i.e. C becomes inactive)
  2) Some edge e goes tight,
    Add e to F and set $\gamma(C_1 \cup C_2) \leftarrow \gamma(C_1) + \gamma(C_2)$ where C_1, C_2 are the components of F bridged by e
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_1 \cup C_2\}$
end while
Let C_r denote the root component of F and $F(C_r)$ denote its edges
while some $e \in F(C_r)$ is such that the new nonroot component C' of $F - \{e\}$ has $C' \in \mathcal{C}$ and $\gamma(C') = 0$ **do**
  $F \leftarrow F - \{e\}$
  Let C_r denote the new nonroot component of this edge set F
end while
Delete every edge e from F that is not in the root component of F.
return (F, P) where P is the set of nodes not in the root component of F

The first loop iterates a polynomial number of times: there can be at most n "deactivations" of an active component between iterations where an edge is added to F. This also means the number of dual variables y_C with positive value constructed is also at most polynomial, so we can implement this algorithm in polynomial time by only keeping track of these dual variables.

Now consider the set $\mathcal{C}$ constructed during the execution of Algorithm 2. Note that $y_S > 0$ means $S \in \mathcal{C}$.

Claim 1 *For any $C, C' \in \mathcal{C}$ we have $C \cap C' = \emptyset, C \subseteq C'$ or $C' \subseteq C$ (i.e. the sets with positive dual form a* laminar family*).*

Proof. Suppose C and C' share a vertex v. Then cannot be that C and C' are different components in some iteration. Say C is a component during iteration i and C' is a component during iteration i' and say that $i < i'$. An easy invariant of this loop is that the components of one iteration are subsets of components of subsequent iterations. So, in iteration i' we have that C is a subset of a component in F. Since C contains $v \in C'$ in iteration i', then it must be that $C \subseteq C'$. ■

Now let (F, P) denote the returned solution. We partition the sets with positive dual into two sets

- $\mathcal{S}_1 = \{C \in \mathcal{C} :$ and $C \not\subseteq P\}$
- $\mathcal{S}_2 = \{C \in \mathcal{C} :$ and $C \subseteq P\}$

In fact, the dual variables for sets in $\mathcal{S}_2$ pay for discarding the nodes in P perfectly.

Lemma 8 $\pi(P) = \sum_{C \in \mathcal{S}_2} y_C$

Proof. We first establish the following loop invariant for the first while loop: for each component C of F (active or not), $\sum_{S \subseteq C} y_S + \gamma(C) = \pi(C)$. Initially this is true because $y = 0$ and $\gamma(\{v\}) \leftarrow \pi(C)$. It holds when raising y_C and decreasing $\gamma(C)$ because it is just transferring value from $\gamma(C)$ to y_C.

Finally, we show it holds when some edge e bridging C_1 and C_2 is added to F. First, note that any $C \in \mathcal{C}$ with $C \subseteq C_1 \cup C_2$ must be either a subset of C_1 or a subset of C_2 by Claim 1. Thus,

$$\sum_{S \subseteq C_1 \cup C_2} y_S = \sum_{S \subseteq C_1} y_S + \sum_{S \subseteq C_2} y_S = \pi(C_1) - \gamma(C_1) + \pi(C_2) - \gamma(C_2) = \pi(C_1 \cup C_2) - \gamma(C_1 \cup C_2).$$

Now consider any $v \in P$. If v was not in the root component of F just after the first while loop (before the pruning), then it lies in a component $C \in \mathcal{C}$ such that $\gamma(C) = 0$. If v was in the root component of F after the first loop but was pruned in the second loop, then it also lies in some $C \in \mathcal{C}$ with $C \subseteq P$ and such that $\gamma(C) = 0$, so $\sum_{S \subseteq C} y_S = \pi(C)$.

Let $C_1, \ldots, C_k$ be the maximal subsets of $\mathcal{S}_2$ (i.e. $C_i \in \mathcal{S}_2$ but there is no $C' \in \mathcal{S}_2$ such that $C_i \subsetneq C'$). We just showed that $P = \cup_{i=1}^k C_i$ and that $\pi(C_i) = \sum_{S \subseteq C_i} y_S$. By Claim 1, any $S \in \mathcal{S}_2$ must be a subset of some C_i so in fact $\sum_{S \in \mathcal{S}_2} y_S = \pi(P)$. ■

Lemma 9 $c(F) \leq 2 \cdot \sum_{S \in \mathcal{S}_1} y_S$

Proof. We only sketch this one since it is similar to an argument we saw in an earlier lecture.

Let F denote the final set of returned edges. Note that any $C \in \mathcal{C}$ with $\delta(C) \cap F$ must satisfy $C \in \mathcal{S}_1$. This is simply because no edge of F has an endpoint in P.

Consider some iteration of the first while loop and let F_i denote the set of edges in this iteration. Consider the graph H that consists of a vertex for each component of F_i and edges $e \in F$ that bridge two of these components. This graph H consists of isolated nodes plus a single tree that includes the root component of F_i. Furthermore, each leaf of H must either be the root component or an active component of F_i in this iteration, otherwise we would have pruned its parent edge in the second while loop.

Therefore, all leaves of H except, perhaps, the root component are active components of F_i meaning the active components/nodes in H have average degree at most 2. Then essentially the same "averaging argument" using relaxed complementary slackness as was used in our Steiner Forest discussion shows that $c(F) \leq \cdot \sum_{S \in \mathcal{S}_1} y_S$. ■

Combining Lemma 8 and 9 shows

$$c(F) + 2 \cdot \pi(P) \leq 2 \cdot \sum_{\emptyset \subsetneq S \subseteq V} y_S \leq 2 \cdot OPT_{PCST},$$

which concludes the proof of Theorem 2.

26.3.2 A Slight Improvement

To get the 5-approximation for k-MST (not just the $5 + \epsilon$ approximation) we needed the slightly stronger statement that $c(F) + (2 - 1/n) \cdot \pi(S) \leq (2 - 1/n) \cdot OPT_{PCST}$. This follows from a slightly stronger degree counting argument.

If T is a tree and S is a subset of nodes of T that includes all but at most one leaf (e.g. the root component in H in the proof of Lemma 9) then one easily show that the average degree of nodes in S is at most $2 - 1/n$.

26.4 Discussion

The 5-approximation presented here is by Chudak, Roughgarden, and Williamson [CRW01] and the current best approximation is a 2-approximation by Garg [G05]. The primal-dual approximation for PRIZE-COLLECTING STEINER FOREST is due to Goemans and Williamson [GW95].

References

CRW01 F. A. Chudak, T. Roughgarden, and D. P. Williamson, Approximate k-MSTs and k-Steiner trees via the primal dual method and Lagrangean relaxation, In Proceedings of IPCO, 2011.

G05 N. Garg, Saving an epsilon: a 2-approximation for the k-MST problem in graphs, In Proceedings of STOC, 2005

GW95 M. X. Goemans and D. P. Williamson A general approximation technique for constrained forest problems, SIAM Journal on Computing, 24:296–317, 1995.