601.435/635 Approximation Algorithms **Lecturer:** Michael Dinitz
Topic: Primal-Dual: Steiner Forest **Date:** 4/9/24
Scribe: Michael Dinitz

21.1 Introduction

Today we're going to go back to the world of network design (which we haven't looked at since Group Steiner Tree) and study a problem known as Steiner Forest, or what the book calls Generalized Steiner Tree. The relationship between Steiner Forest and Steiner Tree is in some sense like the relationship between Multicut and Multiway Cut, as we will see. More importantly, Steiner Forest is one of the most famous and classical examples of a primal-dual algorithm, which is what we're going to see today.

21.2 Definitions

Steiner Forest is defined as follows:

- **Input:**
  - Graph $G = (V, E)$
  - Cost function $c : E \to \mathbb{R}^+$
  - Pairs $(s_1, t_1), \cdots, (s_k, t_k)$ of nodes
- **Feasible solution:** $F \subseteq E$ such that (V, F) contains an s_i-t_i path for all $i \in [k]$.
- **Objective:** $\min \sum_{e \in F} c(e)$

The problem gets its name because it is easy to see that the optimal solution is a forest. This can be thought of as the "opposite" of Multicut: we're given a bunch of pairs, and instead of cutting them, we need to connect them.

Definition 21.2.1 *Let $\mathcal{S}_i = \{S \subseteq V : |S \cap \{s_i, t_i\}| = 1\}$. And let $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_i$.*

This gives the following obvious LP relaxation.

$$\begin{aligned} \text{minimize:} \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to:} \quad & \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \in \mathcal{S} \\ & x_e \geq 0 \quad \forall e \in E \end{aligned}$$

When we take the dual of this LP, we get the following.

$$\begin{aligned} \text{maximize:} \quad & \sum_{S \in \mathcal{S}} y_S \\ \text{subject to:} \quad & \sum_{S \in \mathcal{S}: e \in \delta(S)} y_S \leq c(e) \quad \forall e \in E \\ & y_S \geq 0 \qquad \forall S \in \mathcal{S} \end{aligned}$$

21.3 Algorithm

Let's use this LP and its dual to design a primal-dual approximation algorithm. The first question, as always for a primal dual algorithm, is which dual variable(s) should we raise? Taking inspiration from what we developed last time for $s - t$ shortest path, we're going to raise the variables of connected components. But for the first time, we're going to raise multiple dual variables at the same time. With those ideas in mind, the following algorithm should be pretty natural. It is basically due to Agrawal, Klein, and Ravi [AKR95], although they phrased it without duality. The analysis through duality is due to Goemans and Williamson [GW95].

Algorithm 1

```
F_1 = ∅, y⃗ = 0⃗, j = 1.
while F_j not feasible do
    Let C_j = {S ∈ 𝒮 : S component of (V, F_j)} be the components of (V, F_j) that are also sets in 𝒮.
    Increase all y_S : S ∈ C_j uniformly until ∃e_j ∈ δ(S), S ∈ C_j such that constraint for e_j is tight,
    i.e. Σ_{S∈𝒮:e_j∈δ(S)} y_S = c(e_j).
    Let Δ_j be the amount we raised each y_S (for S ∈ C_j)
    F_{j+1} = F_j ∪ {e_j}.
    j = j + 1.
end while
F = F_j
for k = j − 1 to 1 do
    if F \ {e_k} is feasible then
        Remove e_k from F
    end if
end for
return F.
```

21.4 Analysis

21.4.1 Initial Observations

Lemma 21.4.1 *$\vec{y}$ is always dual feasible.*

Proof: $\vec{y} = \vec{0}$ is feasible at the beginning. At each iteration, we will increase y_S until some constraint is tight for $e \in E$. And such e will be inside the component in the following iterations so its dual constraint will remain tight (not violated). ■

Lemma 21.4.2 *This algorithm is polytime.*

Proof: There are at most $|E|$ iterations and at most n active components in each iteration. So there are at most $n|E|$ nonzero dual variables, so we can just keep track of them (even though the total number of dual variables is exponential). And each iteration can be computed in polynomial time by directly computing the amount we can raise the active dual variables until a constraint becomes tight (rather than the more continuous way the algorithm was presented) ■

Observation 21.4.3 *Final pruning is necessary.*

Proof: Consider the star graph where s_1 is in the center connected to $v_1, \cdots, v_{n-2}$ with costs all 1 and connected to t_1 with cost 3. Then without the final pruning, the algorithm would buy the entire star rather than just the $\{s_1, t_1\}$ edge. ■

21.4.2 Main Analysis

For the rest of the lecture, we're going to focus on proving the following theorem:

Theorem 21.4.4 *The Primal-Dual algorithm is a 2-approximation.*

In order to prove this, the main technical tool that we're going to use is the following lemma:

Lemma 21.4.5 $\sum_{S \in \mathcal{C}_j} |F \cap \delta(S)| \leq 2|\mathcal{C}_j|$ *for all iterations j (note that F is the* final *F, not any F_j).*

Before we prove this lemma, let's see how it implies the theorem. First, note that

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S \in \mathcal{S}: e \in \delta(S)} y_S = \sum_{S \in \mathcal{S}} |\delta(S) \cap F| y_S$$

To analyze this, let's prove another claim.

Claim 21.4.6

$$\sum_{S \in \mathcal{S}} |\delta(S) \cap F| y_S \leq 2 \sum_{S \in \mathcal{S}} y_S \tag{21.4.1}$$

Proof: Let's do induction on the iterations of the algorithm. Initially, since $y = \vec{0}$ both sides of (21.4.1) are equal to 0. Now consider some iteration j, and suppose that (21.4.1) is true at the beginning of the iteration. Let Δ_j denote the amount that we increased the active dual variables (the moats) at this iteration. Then the left hand side of (21.4.1) increases by

$$\sum_{S \in \mathcal{C}_j} |\delta(S) \cap F| \Delta_j \leq 2|\mathcal{C}_j| \Delta_j,$$

where the inequality is from Lemma 21.4.5.

On the other hand, the right hand side of (21.4.1) increase by exactly $2|\mathcal{C}_j|\Delta_j$. Thus by induction, (21.4.1) continues to hold. ■

Thus we have that

$$\sum_{e \in F} c(e) \leq 2 \sum_{S \in \mathcal{S}} y_S \leq 2 \cdot \mathrm{OPT},$$

where the final inequality is from weak duality.

21.4.2.1 Proof of Lemma 21.4.5

A simple induction implies that F_j is a forest for all j. Now fix some time j, and consider the graph $G_j = (V_j, E_j)$ defined as follows:

- V_j has a vertex for each connected component of (V, F_j), and
- $E_j = \{\{S, T\} : \exists u \in S, v \in T \text{ s.t. } \{u, v\} \in F\}$.

Note that every edge in E_j corresponds to *exactly* one edge in F (the existential is satisfied by exactly one edge), since otherwise there would be a cycle in F. Recall that $\mathcal{C}_j \subseteq V_j$ are the set of active components at time j, i.e., the connected components that are also in $\mathcal{S}$. So another way of stating Lemma 21.4.5 is that the average degree of nodes in $\mathcal{C}_j$ in G_j is at most 2.

So let's think about $\mathcal{C}_j$ and G_j. First, clearly G_j is a forest, since it can be constructed by starting with a forest (F before the pruning) and then contracting all edges in F_j and deleting all edges that were removed by the pruning.

Lemma 21.4.7 *Let $S \in V_j$ have degree 1 in G_j. Then $S \in \mathcal{C}_j$.*

Proof: Suppose that $S \notin \mathcal{C}_j$, and let e be the unique edge incident on S in G_j. Since $S \notin \mathcal{C}_j$ but is in V_j, by definition $S \notin \mathcal{S}$. Thus S does not separate any s_i from t_i, so in the final pruning we would have removed e from F. This is a contradiction, since $e \in F$. Thus $S \in \mathcal{C}_j$. ■

So consider the forest G_j. For every tree in G_j, all the leaves are in $\mathcal{C}_j$. It turns out to be a fact about trees that if you include all of the leaves, the average degree is small.

Lemma 21.4.8 *Let T be a tree. If $S \subseteq V(T)$ contains all leaves in T, then $\sum_{v \in S} d(v) \leq 2|S|$.*

Proof:

$$\begin{aligned}
\sum_{v \in S} d(v) &= \sum_{v \in V(T)} d(v) - \sum_{v \notin S} d(v) = 2(|V(T)| - 1) - \sum_{v \notin S} d(v) && \text{(number of edges in a tree)} \\
&\leq 2(|V(T)| - 1) - 2(|V(T)| - |S|) && \text{(all non-}S\text{ nodes are non-leaves)} \\
&= 2|S| - 2 \leq 2|S|
\end{aligned}$$

as claimed. ■

Lemmas 21.4.7 and 21.4.8 imply that the average degree of nodes in $\mathcal{C}_j$ is at most 2 in G_j, which implies Lemma 21.4.5. Slightly more formally, we have

$$\sum_{S \in \mathcal{C}_j} |F \cap \delta(S)| = \sum_{S \in \mathcal{C}_j} d_{G_j}(S) \leq 2|\mathcal{C}_j|,$$

where $d_{G_j}(S)$ denotes the degree in G_j of S (which is a single node in G_j). This completes the proof of Lemma 21.4.5, and thus the proof of Theorem 21.4.4

21.5 Extensions and Open Questions

The approximability of Steiner Forest is still open, and is one of the most fundamental network design problems for which there are still gaps. In particular, this 2-approximation is still the best known approximation ratio, while the only hardness result we have is that it is APX-hard (and thus there is *some* constant $c > 1$ such that it cannot be approximated to ratio better than c). But whether it is possible to do better than 2 is still an important open question. Note that for Steiner Tree there is a very easy 2-approximation (compute an MST on the metric closure), but with more work it is possible to do better. Is Steiner Forest actually harder than Steiner Tree? We still don't know.

There are numerous extensions of Steiner Forest which make the problem far more difficult. For example, in the Steiner k-Forest problem we are also given a parameter k, and are asked to connect at least k pairs rather than all of them. The best-known algorithm for this is an $O(\sqrt{n})$-approximation [GHNR10]. For the special case where $c(e) = 1$ for all $e \in E$, we can do slightly better: Guy Kortsarz, Zeev Nutov, and I gave an $O(n^{\frac{1}{3}(7-4\sqrt{2})}) \approx O(n^{0.44772})$-approximation [DKN17].

References

[AKR95] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[DKN17] Michael Dinitz, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithm for steiner k-forest with nearly uniform weights. *ACM Trans. Algorithms*, 13(3):40:1–40:16, 2017.

[GHNR10] Anupam Gupta, Mohammad Taghi Hajiaghayi, Viswanath Nagarajan, and R. Ravi. Dial a ride from k-forest. *ACM Trans. Algorithms*, 6(2):41:1–41:21, 2010.

[GW95] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.