Lecture 1

CS522: Advanced Algorithms

October 18, 2004
Lecturer: Kamal Jain
Notes: Ethan Phelps-Goodman

1.1 The Min Cost Steiner Forest Problem

In the last lecture we saw primal-dual schemas for weighted vertex cover and facility location. In this lecture we will look at a slightly more complicated primal-dual schema algorithm.

Problem 1 (Steiner Forest). *We are given an undirected graph $G = (V, E)$ with edge weights $c_e \geq 0$. We are also given a function $r : V \times V \rightarrow \{0, 1\}$ where*

$$r(u, v) = \begin{cases} 1 & \text{if any feasible solution must contain a path between } u \text{ and } v \\ 0 & \text{otherwise} \end{cases}$$

Our task is to come up with a minimum weight subgraph H of G that satisfies the connectivity constraints given by r.

Note that this is a generalization of the Steiner tree problem where only certain subsets of the Steiner vertices are required to be connected.

Lemma 1. *The min cost Steiner tree is a forest (ie. it contains no cycles.)*

Proof. Were there a cycle we could remove an edge along the cycle and not break any connectivity. This would be a feasible solution of lower cost, contradicting our assumption of optimality. □

1.2 Formulation as a linear program

First a bit of notation. Let us define a function f on subsets of V. f will be 1 for those sets S that define a cut that separates two vertices that must be in the same component.

$$f(S) = \begin{cases} 1 & \text{if } \exists u \in S, v \notin S \text{ s.t. } r(u,v) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We use the notation $\delta_G(S)$ to mean the set of edges that cross the cut given by S.

$\delta_G(S) = \{e \mid e = (u,v)$ is an edge in graph G s.t. $u \in S, v \notin S$ or visa versa $\}$

These edges are known as cross edges.

Note that if there is some set S with $f(S) = 1$ and $|\delta_H(S)| = 0$ then H is not a feasible solution. Conversely, if all subsets S of V with $f(S) = 1$ satisfy $|\delta_H(S)| \geq 1$, then H is a feasible solution. We will use this fact to construct the linear program.

The primal will have a variable x_e for each edge. We will consider $x_e = 1$ to mean $e \in H$, and $x_e = 0$ to mean $e \notin H$. Stated as a linear program, the Steiner forest problem is:

$$\text{minimize } \sum_{e \in E} c_e x_e$$

subject to constraints

$$\forall S \text{ s.t. } f(S) = 1 \quad \sum_{e \in \delta_G(S)} x_e \geq 1$$
$$x_e \geq 0$$

The dual of this program is:

$$\text{maximize } \sum_{S: f(S)=1} y_S$$

subject to

$$\forall e \quad \sum_{S \mid e \in \delta_G(S)} y_S \leq c_e$$
$$y_S \geq 0$$

1.3 The iterative algorithm

As in the facility location algorithm from last lecture, we give a combinatorial algorithm rather than solving the LP directly. Our approach will be to raise a set of dual variables until one of the constraints goes tight. We then pick the corresponding primal variable to be an edge in the solution. Once we obtain a feasible solution we go through a pruning phase to improve the quality of the solution.

We refer to a subset of vertices as a component, labeled C_i. A component is said to be active if $f(C_i) = 1$. Our initial set of components will be sets of single vertices $C_1 \dots C_n$ where each C_i contains the ith vertex of G. Each iteration of the algorithm raises all active components simultaneously until one or more goes tight. We then pick one of the now tight constraints and include the corresponding edge in H. This new edge will connect two components, which become merged into one. The algorithm terminates when there are no active components left.

Lemma 2. *At any point in the algorithm, H is a forest.*

Proof. Once two components are joined the edges inside the merged component won't be considered further. Any future edges will connect this component to outside components, so a cycle is never created. □

Lemma 3. *The algorithm stops at the first feasible solution. In other words, H is a feasible solution if and only if there are no active components.*

Proof. ? □

The algorithm finishes with a pruning phase, which will be important later in the analysis. The pruning phase simply checks for every edge $e \in H$ if removing e maintains a feasible solution. If $H - e$ is feasible, e is removed.

1.4 Proving the approximation ratio

In this section we will prove that the primal-dual algorithm gives a 2-approximation for the Steiner forest problem. To start, note that the primal slackness condition

$$x_e > 0 \implies \sum_{S: e \in \delta_G(S)} y_S = c_e$$

is true since we set $x_e = 1$ precisely when the dual goes tight. The dual slackness condition

$$y_S > 0 \quad \Longrightarrow \quad \sum_{e \in \delta_G(S)} x_e = 1$$

is not true in general. (If it were we'd have an optimal solution.) To prove an α approximation ratio we want to show

$$y_S > 0 \quad \Longrightarrow \quad \sum_{e \in \delta_G(S)} x_e \le \alpha \cdot 1.$$

Unfortunately this is not true for any constant α. It is enough to show however that the bound is true on average. In particular, we will show

$$\sum_S \left(y_S \sum_{e \in \delta(S)} x_e \right) \le \sum_S 2 y_S \tag{1.1}$$

At the start of the algorithm this is trivially true, since all $y_S = 0$ for all S. Now say that over some arbitrary interval the algorithm has raised the active components by Δ. Then the right hand side of (1.1) increases by

$$2\Delta \cdot \; \# \text{ active components.} \tag{1.2}$$

For the left hand side of (1.1), we will need the graph obtained by compressing each active component into a single node. Call this graph H_P. Then $\deg_{H_P}(S) = \delta_H(S)$. Using this notation, over an interval Δ, the left hand side adds

$$\sum_{S_i : S_i \text{ is an active component}} \Delta \cdot \deg_{H_P}(S_i) \tag{1.3}$$

We need to show that (1.3) is always less than (1.2). Note that for any forest we have

$$\sum_{v \in V} \deg(v) < 2 \cdot |V| \tag{1.4}$$

This follows from the fact that the sum of all the degrees is precisely twice the number of edges, and the number of edges of a forest is always less than the number of nodes. In our forest H_P, we can split this into

$$\sum_{S_i \text{ active}} \deg(S_i) + \sum_{S_i \text{ inactive}} \deg(S_i) < 2 \cdot \# \text{ active comp.} + 2 \cdot \# \text{ inactive comp.} \tag{1.5}$$

Claim 1. *An inactive component cannot be a leaf in H_P.*

Proof. Assume that C is an inactive leaf connected to the tree by edge e. Since C is inactive, it doesn't need a path from anything inside the component to the rest of the graph. Therefore in the pruning phase, $H - e$ will still be a feasible solution, so e will be removed, and C will no longer be a leaf. $\square$

This implies the average degree of inactive nodes is at least 2. Taken together with (1.5) we get the desired bound

$$\sum_{S_i active} \deg(S_i) < 2 \cdot \# \text{ active comp.} \tag{1.6}$$

This shows that equation (1.1) always holds, and so our we have proved that the algorithm is a 2-approximation.