COMPSCI 590.1: Graph Algorithms March 31, 2015

Lecture # 23

Lecturer: Debmalya Panigrahi *Scribe: Brandon Fain*

1 Overview

In the last lecture we looked at primal-dual algorithms for approximating the Generalized Steiner Forest problem with edge weights, in which you want to find a minimum cost subgraph which satisfies arbitrary connectivity requirements between pairs of terminals. The problem can be phrased in terms of choosing edges so as to satisfy requirements on all cuts, as is made formal in definition 1. Today, we will again examine the Generalized Steiner Forest problem but this time we will not use any primal-dual scheme. Instead, we will use the technique of iterative rounding as introduced for this problem in [Jai01]. The general idea is that rather than running an LP relaxation of the integer program one time then rounding all results to get a solution, we iteratively run the LP, pick part of the solution, run the LP, pick another part, etc.

2 Generalized Steiner Forest and Iterative Rounding

Let us begin by formalizing the problem. We are given a graph $G = (V, E)$ and edge costs c_e for all edges. We are also given a list of terminal requirements, for any terminal pair u and v we get $r_{u,v}$, the connectivity requirement. We have to find a subgraph $H \subseteq G$ such that for all u, v pairs in G, u, v are connected by $r_{u,v}$ paths in H.

We can rewrite these demands as follows: for every cut S in G, let $f(S)$ be the maximum $r_{u,v}$ with $u \in S$ and $v \notin S$ (that is, $f(S)$ is the largest requirement accross the cut). Then we can write the problem as an integer linear program with variables $x_e \in \{0, 1\}$ indicating whether edge e is chosen to be in H. We relax this constraint so that $x_e \in [0, 1]$ to get the following linear program:

Definition 1. *We define LP_{GSF} to be the following:*

$$\begin{array}{lllll} \textit{Minimize} & \sum_{e \in E} & c_e x_e & & \\ & \sum_{e \in S} & x_e \geq f(S) & & \forall S \subset V \\ & 0 & \leq x_e \leq 1 & & \forall e \in E \end{array}$$

This is the same linear program that previous primal-dual techniques to solve the problem used. However, these techniques all had approximations that depended on the value of r_{max}, the largest connectivity requirement. In 1998, Jain developed a new approach to the problem called iterative rounding that achieves a 2-approximation to the problem. Moreover, the algorithm is quite simply to describe, it simply iteratively solves the LP, picks a single edge to either put in H or throw out entirely, and solves the LP again, continuing in this manner until it has a feasible solution.

The one note of caution here is that there are an exponential number of cut constraints. This is not however a problem, as we have a separating oracle for this particular problem. A separating oracle is a subroutine that, given a point in the solution space $\mathbf{x}$ either returns that it is feasible (meaning that $\mathbf{x}$ lies in the constraint space) or returns a hyperplane separating $\mathbf{x}$ from the constraint space. The existence of such a

Algorithm 1 Generalized Steiner Forest Algorithm

```
1: procedure GSF(G)
2:     F ← ∅
3:     repeat
4:         Run LP_GSF to get x*
5:         if there is an x*_e = 0 then
6:             Delete e from G
7:         else
8:             if there is an x*_e = 1 then
9:                 F ← F ∪ e
10:            else
11:                if there is an x*_e ≥ 1/2 then
12:                    F ← F ∪ e
13:                end if
14:            end if
15:        end if
16:        Modify f(S) to reflect the change
17:        Rewrite LP_GSF to reflect the change
18:    until All requirements are satisfied
19: end procedure
```

separating oracle allows us to still solve the LP efficiently using ellipsoid methods even with an exponential number of constraints.

Correctness and runtime are both trivial arguments if we assume that one of the three if conditions always hold, that is, if the algorithm never gets stuck. We will show this later, but for now assume it is true and note that the algorithm then simply adds edges until it satisfies all requirements and completes after solving a polynomial number of LPs. The main result is the approximation factor:

Theorem 1. *Algorithm 1 gives a 2-approximation.*

To prove the approximation factor, we need two things. The first and easier is to show that if the algorithm never gets stuck (that is, for every run of the LP, there is always some x_e equal to 0 or 1 or greater equal to 1/2), then it gives a 2-approximation. Then, we will spend most of the rest of our focus proving that indeed the algorithm never gets stuck.

Lemma 2. *If Algorithm 1 never gets stuck then it is a 2-approximation.*

Proof. Let $LP(\mathbf{x_0^*})$ be the first solution to the LP, where $LP(\mathbf{x_0^*}) \leq OPT$, for OPT the value of the cost of the optimal integral solution. Then, consider note that $LP(\mathbf{x_1^*}) \leq LP(\mathbf{x_0^*}) - c_{e_0}x^*_{e_0}$ where $\mathbf{x_1^*}$ is the solution to the second LP, c_{e_0} is the cost of the edge selected to include from the first solution to the LP, and $x^*_{e_0}$ is the value of that edge in that solution to the LP. We can get this bound because after buying edge e_0 in the first iteration, the solution to the first LP is still a feasible solution for the second LP with cost decreased by at least the amount of this edge we bought. But, note that this reasoning continues ad nauseum for all further LPs. This yields the following telescoping sum:

$$LP(\mathbf{x_1^*}) + LP(\mathbf{x_2^*}) + \ldots + 0 \;\leq\; LP(\mathbf{x_0^*}) - c_{e_0}x^*_{e_0} + LP(\mathbf{x_1^*}) - c_{e_1}x^*_{e_1} + \ldots + LP(\mathbf{x^*_{q-1}}) - c_{e_{q-1}}x^*_{e_{q-1}}$$

where there are q total edges in the solution. Most terms cancel, giving:

$$\sum_{i=0}^{q-1} c_{e_i} x_{e_i} \leq LP(\mathbf{x_0^*}) \leq OPT$$

However, note that the algorithm pays $\sum_{i=0}^{q-1} c_{e_i}$ which is at worst twice the cost of $\sum_{i=0}^{q-1} c_{e_i}$. Thus, we get the 2-approximation. □

Now we need to prove the more difficult claim: that the algorithm never gets stuck, it can always find an edge that satisfies one of the three ifs.

Lemma 3. *In every iteration of Algorithm 1, there is at least one edge e with $x_e = 0, x_e = 1$, or $x_e \geq 1/2$ in the LP solution.*

Proof. Suppose by contradiction that all edges have values $x_e \in (0, 1/2)$ in the LP solution for some iteration of the algorithm. We want to show this leads to a contradiction using a token counting argument. To do this, we need another lemma about tight and laminar sets which we do not prove (it was a previously known result of sets); here a tight set is all constraints are tight for that set, that is a set S is tight for solution X if $X(S) = f(S)$.

Lemma 4. *There exists a set L of m tight sets that have the properties:*

1. *The sets are laminar ($\forall A, B \in L$, $A \subseteq B$ or $A \cap B = \emptyset$).*
2. *The incidence vector of edges in set S are linearly independent.*

Given this lemma, we begin a token counting argument using these m tight sets L. Give every edge a total token value of 1. Then, edge $e = (u, v)$ gives x_e to the smallest set in L containing u and the smallest set in L containing v (which does not exceed 1 by the contradictory assumption). Then, give $1 - 2x_e$ tokens to the smallest set in L containing u and v. In order to show this leads to a contradiction, we prove the following two properties:

1. $\forall S \in L$, S gets ≥ 1 tokens.
2. There are more than 0 tokens remaining.

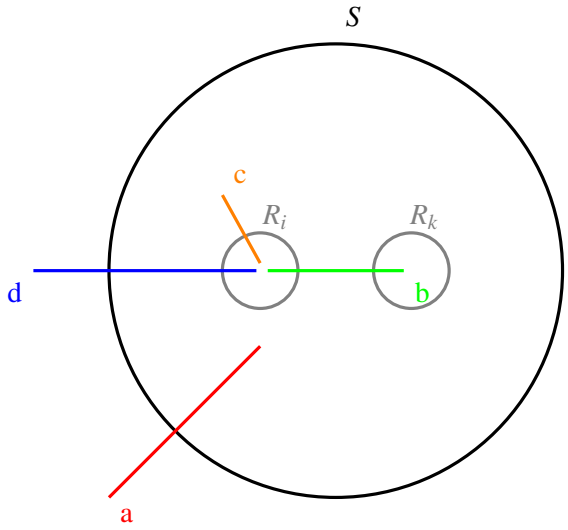
Note that these properties immediately imply that there were more than m tokens to begin with, which is a contradiction of our construction giving each edge 1 token.

We begin by proving property 2. Consider some maximal $S \in L$. Since we assumed that $x_e < 1/2$, it must have at least 3 cut edges. This is because S is a tight set, meaning that the constraint that $\sum_{e \in S} x_e \geq f(S) \geq 1$ which requires at least three edges to contribute. Furthermore, since we took the maximal laminar set, no other set in L contains S, and thus no set in L contains both endpoints of these three edges. Thus, the $1 - 2x_e$ amount that the edges contribute goes unused, so there is some amount of tokens greater than 0 going unused, proving property 2.

Now we prove property 1. Let $T(S)$ be the tokens that this set receives. We want to show that $T(S)$ is an integer and is positive, which implies property 1, since S is the maximal set in L.

Let $R_1, ..., R_k$ be maximal sets in S. Consider all possible edges from some R_i as in the following figure:

Figure 1: Edges from R_i



By linear independence, at least some a, b, or c edges must exist. Also, the set S gets token revenue equal to $(b+-2x_b)+(c-x_c)+x_a$ where b,c just count number of edges and thus are clearly integer. However, note that $-2x_b-x_c+x_a=(-1)(2x_b+x_c-x_a)=(2x_b+c_x+x_d)-(x_a+x_d)$. The first term is just $\sum_i f(R_i)$ and the second term is just $f(S)$, both of which must be integer terms because S is a tight set. Thus, all of these terms, and therefore $T(S)$, must be integer, and clearly greater than 0, implying property 1, that every such set S gets at least 1 token.

Therefore, we conclude that there are more than m tokens in total, which is a contradiction of our assumption that in some run of the LP, all values x_e are in $(0,1/2)$. So, the algorithm never gets "stuck."

□

3 Summary

In this lecture we have used the technique of iterative rounding as introduced by Jain in [Jai01] to solve the generalized steiner forest problem with edge weights. The algorithm is a 2-approximation and has a worst case run time of $O(m)$ times the time to solve the LP at each step. The general and novel idea is that rather than obtaining an LP relaxation of the integer program one time then rounding all results to get a solution, we iteratively solve the LP, pick part of the solution, run the LP, pick another part, etc.

References

[Jai01] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.