College of Computer & Information Science — Fall 2007
Northeastern University — 16 October 2007
CS G399: Algorithmic Power Tools I — Scribe: Hooman Javaheri

Lecture Outline:

- Generalized Steiner Network
- Jain's Algorithm (2-Approximation)

In this lecture, we will study a generalized form of the Steiner network problem. We will present a 2-approximation algorithm for this problem due to Kamal Jain.

1 Generalized Steiner Network

In the previous class we covered the Steiner Forest problem. A generalization of this problem is the following:

Problem 1. Given a graph $G(V, E)$, a cost function $c : E \to Z^+$, and a connectivity requirement r_{uv} for each pair (u, v), find a min-cost subgraph of G satisfying the connectivity requirement for all (u, v), where u_e is maximum number of copies you can pick for edge e

We can write the problem as the following linear program:

$$\min \sum c_e x_e$$

$$s.t. \sum_{e \in \delta(S)} x_e \geq f(S)$$

$$0 \leq x_e \leq u_e,$$

where

$$f(S) = \max_{\substack{u,v \\ u \in S, v \in \bar{S}}} r_{uv}$$

$\delta(S)$ is the set of edges connecting S to $\overline{S}$.

2 Jain's algorithm for generalized steiner network problem

Jain's algorithm is a 2-approximation algorithm for Generalized Steiner Network. This algorithm use the following approach to solve the problem: Solve the LP, construct part of the solution by selecting edges for which x_e is large, redefine a new LP, and repeat until the whole connectivity requirements are satisfied. The algorithm is formally specified in Algorithm 1.

This algorithm looks neat, but here are some issues:

Algorithm 1: Jain's algorithm for generalized steiner network problem

1. $F \leftarrow \emptyset$, Define f according to r_{uv}s. $f' \leftarrow f$.
2. **repeat**
   - 2.1. Solve LP for f' to obtain a solution x with desired property: $\exists e \;\; s.t. \;\; x_e \geq 1/2$
   - 2.2. Add $\lceil x_e \rceil$ copies of all e s.t. $x_e \geq 1/2$ to F
   - 2.3. Remove the above edges e from G.
   - 2.4. $f'(S) \leftarrow \max(0, f(S) - \delta_F(S))$ where, $\delta_F(S)$ is set of edges of F crossing S.

   until $f'(S) = 0$
3. Return F

- How do we solve the LP efficiently? Looking at the definition, the number of the constraints in the LP is exponential. To obtain a poly-time algorithm we need to solve LP in poly-time. Can we do that?
- It might be the case that in some iteration the redefined LP cannot be solved (infeasible).
- How do we calculate $f^{'}(S)$ in poly-time? It's easy to see that we have exponential number of the sets in each iteration.
- Does the algorithm terminate? Is the termination time polynomial?

In order to prove that this algorithm is a 2-approximation algorithm, we need to answer above questions. Our approach to answer the questions is as following. We will prove:

1. The LP can be solved in polynomial time; in fact, we will find an optimal BFS in each iteration. And we will solve the LP without explicitly maintaining the function $f(\cdot)$.
2. The algorithm is a 2-approximation assuming the desired property in statement 2.1 is true.
3. Desired Property: for all BFSs, $\exists e \;\; s.t. \;\; x_e \geq 1/2$ Actually, we will prove it for $1/3$ rather than $1/2$. The approach for $1/2$ is the similar, but requires some complicated case analysis, for which we refer to the original paper.

These proofs answer the challenges we face. Part 1 answers the issue about solving the LP. Parts 1 and 2 together prove the termination of the algorithm. Looking at step 2.3 in algorithm, we can easily see if we find BFS with desired property, at least one edge will be removed from our graph, so the termination time of the algorithm will be polynomial. The most important part of the proof is establishing the desired property of step 2.1.

Claim 1. *We can solve the LP in polynomial time.*

Proof. We know our LP has exponential number of constraints, therefore it is inefficient to solve this LP going through all constraints. Our proof is based on the **Seperation Oracle Method.**

Theorem 1 (Grötschel, Lovász, Schrijver). *An LP*

$$\min C^T x$$

$$s.t. \;\; x \in P$$

which has exponential number of constraints can be solved to yield an optimal BFS in polynomial time given a polynomial time procedure (oracle) that determines for a given x, either $x \in P$ or a hyperplane (e.g. violating constraint) which separates x from P.

All we need to show is our LP has this property. Let's look at the LP again.

$$\min \sum C_e x_e$$

$$s.t. \sum_{e \in \delta(S)} (x_e) \geq f(S)$$

$$0 \leq x_e \leq u_e$$

where,

$$f(S) = \max_{\substack{u,v \\ u \in S, v \in \bar{S}}} r_{uv}$$

It's easy to see that any solution for LP satisfies all cut constraints. That means x allows a flow of size r_{uv} from u to v. Now we can define our polynomial-time oracle.

Given x, set the x_e to be capacity of edge e. Run **MAX-FLOW** from u to v. If flow $\leq r_{uv}$, return MIN-CUT separating u and v which has *capacity* $< r_{uv}$. If $\forall u, v\, flow \geq r_{uv}$ say x is feasible ($x \in P$).

We know MAX-FLOW runs in Poly-time. To solve our LP we need $O(n^2)$ MAX-FLOW executions totally. (This number can be improved to $n-1$ using Gomory-Hu trees.) So in the first iteration we can solve our LP in poly-time.

Now suppose we are in the second iteration. We have already selected some edges and added them to our solution F. The second iteration gives us x' as a new solution according to f'. We can solve LP defined by f' by running $O(n^2)$ MAX-FLOW computation on the graph given by $x' + F$. The oracle is defined as before. If the oracle confirms that $x' + F$ is feasible, we can make sure that in each iteration our accumulative solution would be feasible for the original problem.

We can solve this LP using the separation oracle method, which in fact, yields an optimal BFS in poly-time. □

Theorem 2. *Jain's algorithm is a 2-approximation algorithm assuming we can always find an optimal BFS satisfying desired property of step 2.1.*

Proof. **(Proof by induction on number of iterations)**
Base case: Suppose we have only one iteration in our algorithm. (i.e. the algorithm give the final solution after one iteration.)

$$Cost(F) = \sum_{e \in F, x_e \geq 1/2} c_e \cdot \lceil x_e \rceil$$

$$\leq 2 \cdot \sum_{e \in F, x_e \geq 1/2} c_e \cdot x_e$$

$$\leq \sum_{e \in F} c_e \cdot x_e$$

$$= 2LP_{\text{OPT}} \leq 2 \cdot \text{OPT}$$

Suppose for a given f', our solution F' is obtained in t iterations and has $Cost \leq 2 \cdot LP_{\text{OPT}}(f')$. We are solving the problem for f. Suppose, in the first iteration, solution returned is x.

$$LP_{\text{OPT}} = \sum_e c_e \cdot x_e$$

F_1 = edges picked in the first iteration ($\lceil x_e \rceil$ copies of each edge with $x_e \geq 1/2$)

$$f'(S) = f(S) - \delta_{F_1}(S)$$

Solving LP for f', we get solution F' and our final solution will be $F_1 \bigcup F'$.

$$Cost(F) \leq Cost(F_1) + Cost(F')$$

$$\leq \sum_{e \in F, x_e \geq 1/2} c_e \cdot \lceil x_e \rceil + 2 \cdot LP_{\text{OPT}}(f')$$

we need to prove that

$$Cost(f) \leq 2 \cdot LP_{\text{OPT}}(f),$$

which holds true if

$$LP_{\text{OPT}}(f') \leq LP_{\text{OPT}}(f) - \sum_{e \in F, x_e \geq 1/2} c_e . x_e$$

Define

$$\tilde{x} = \begin{cases} \lceil x_e \rceil & x_e > 1/2 \\ 0 & \text{o/w} \end{cases}$$

$$x - \tilde{x} = \begin{cases} x_e & x_e < 1/2 \\ 0 & x_e \geq 1/2 \end{cases}$$

$x - \tilde{x}$ is a feasible solution for LP(f'), so we can say

$$LP_{\text{OPT}}(f') \leq LP_{\text{OPT}}(f) - \sum_{e : x_e \geq 1/2} c_e \cdot x_e$$

and we are done.

□

Theorem 3. *For every BFS for the generalized steiner network problem's LP,* $\exists e$ *s.t.* $x_e \geq 1/2$.

Note that this theorem has nothing to do with objective function and only concerns the polytope constructed by the set of constraints. We will prove this theorem in two steps. First, we will establish the following theorem.

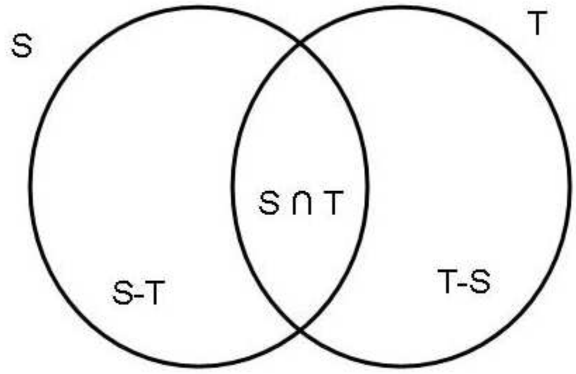

Figure 1: Definition of crossing sets

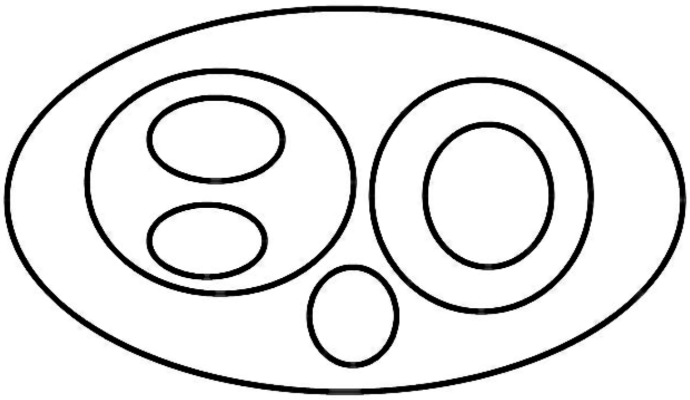

Figure 2: Example of a Laminar family

Theorem 4. *Assume $x_e \in (0,1)$ (i.e. not including 0 and 1) for all e. Suppose there are m such edges. There exist a set of m "TIGHT" constraints that are independent and form a laminar family.*

By independent, we mean none of the constraints can be written as linear combination of others. A laminar family is defined as follows:

Definition 1: Two sets S and T cross if $S - T, T - S, S \cap T$ are all non-empty. Look at Figure 1.

Definition 2: A laminar family is a collection of sets no two of which cross. Figure 2 shows an example of a laminar family. In some sense, a laminar family looks like a hierarchy of sets. (This observation helps us to prove the theorem later on.)

Using the above theorem, we will then establish the desired property.

We will discuss more about laminar families and complete the proof next class.