

Lecture Outline:

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

In this lecture, we will go through the generalized form of Steiner network problem. We will prove Jain's Algorithm, A 2-Approximation algorithms for this form.

1 Generalized Steiner Network

in the previous classes we went through the steiner forest problem. A generalized form of this problem can be viewed as following:

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

We can write the problem down as following liner program:

$$\begin{aligned} \min \quad & \sum C_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S) \\ & 0 \leq x_e \leq u_e \end{aligned}$$

where,

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

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

2 Jain's algorithm for generalized steiner network problem

Jain's algorithm is an 2-approximation algorithm for generalized steiner network. This algortihm use this approach to solve the problem: Solving LP, construct part of the solution and redefine a new LP until the whole connectivity requirements get satisfied.

More formally, we can write the algorithm as 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 S with desired property: $\exists e \text{ s.t. } x_e \geq 1/2$
 - 2.2. Add $\lceil x_e \rceil$ copies of all e s.t. $x_e \geq 1/2$ to F
 - 2.3. Remove 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 to solve LP? 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?
- There might be the case that in some iteration the redefined LP cannot be solved (infeasible).
- How to 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 want to prove that:

1. LP can be solved in Poly-Time and we can also find BFS in each iteration.
2. Algorithm is 2-approximation assuming desired property in state 2.1 is TRUE.
3. Desired Property: for all BFSs, $\exists e \text{ s.t. } x_e \geq 1/2$ Actually, we will prove it for $1/3$ rather than $1/2$. The approach for $1/2$ would be the similar plus some complicated case analysis which is not in our class scope.

These proofs can answer the issues we faced. Part 1 answers the issue about solving LP. Part 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 Poly(E).

The most important part of the proof is part C (desired property). we will go through a complete proof.

Claim 1. *We can solve LP in Poly-time.*

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

Theorem 1. *An LP*

$$\begin{aligned} &\min C^T x \\ &\text{s.t. } x \in P \end{aligned}$$

which has exponential number of constraints can be solved (and also BFS can be yield) in Poly-Time if \exists a Poly-Time procedure (oracle) that determines for a given x , either $x \in P$ or gives a hyperplane (e.g. violating constraint) which separate x from P .

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

$$\begin{aligned} & \min \sum C_e x_e \\ & s.t. \sum_{e \in \delta(S)} (x_e) \geq f(S) \\ & 0 \leq x_e \leq u_e \end{aligned}$$

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 (Look at definition of $f(S)$ again). That means x allows a flow of size r_{uv} from u to v . Now we can define our Poly-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 totally. (This number could be degraded to $n - 1$ using Gomory-Hutrets method.)

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 . second iteration give 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 performs as before. If oracle confirm that $x' + F$ is feasible, we can make sure that in each iteration our accumulative solution would be feasible for the original problem.

All things considered, we can solve this LP and also yield BFS in Poly-Time.

□

Theorem 2. *Jain's algorithm is 2-Approximation algorithm assuming we can always find BFS satisfying desired property of part 3.*

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.)

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

$$\begin{aligned}
&\leq \sum_{e \in F} c_e \cdot x_e \\
&= 2LP_{\text{OPT}} \leq 2.OPT
\end{aligned}$$

Suppose for a given f' , our solution F' is obtained in t iterations and has $Cost \leq 2.LP_{\text{OPT}}(f')$. 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 $F_1 \cup F'$.

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

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

we need to prove that

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

this holds true if

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

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 all BFS for generalized steiner network problem $LP, \exists e$ s.t. $x_e \geq 1/2$.

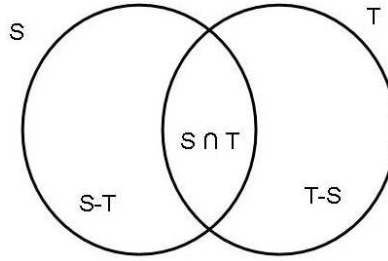


Figure 1: Example of a Laminar family

As a side note you might have noticed this theorem has nothing to do with objective function and is only about "Polytope" constructed by set of constraints.

We will proof this theorem in two steps:

1.

Theorem 4. *Suppose x is a BFS of following LP*

$$\begin{aligned} \min \quad & \sum C_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S) \\ & 0 \leq x_e \leq u_e \end{aligned}$$

Further assume $x_e \in (0, 1)$ not including 0 and 1. Suppose there are m such edges. There exist a set of m "TIGHT" constraints S that are independent and form a laminar family.

2. if 1 then, $\exists e$ s.t. $x_e \geq 1/2$.

By independent, we mean none of the constraints can be written as combination of others. Moreover, One may ask this question about laminar family. Here is the definition of Laminar family:

Definition 1: Two sets S and T crossed 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 fact help us to prove the theorem later on).

We will discuss more about laminar families and complete the proof of part 3 in the next class.

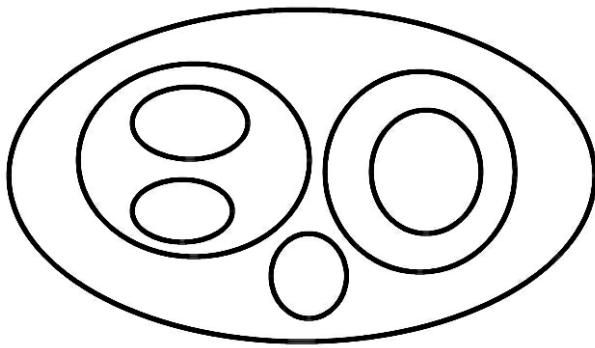


Figure 2: Example of a Laminar family