Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 06

Lecture 27

Lecture 27 : 3 Factor Approximation Algorithm for Prize Collecting Steiner Tree

Welcome in this lecture we will study the price collecting standard tree problem and we will see a constant factor approximation algorithm for it and we will use the technique of deterministic rounding of linear programs. So, let us begin. So, we are continuing the algorithm design framework of deterministic rounding of linear programs, linear programming relaxation and our today's problem is price collecting standard tree. Before that let us recall what is the classical Steiner tree problem. We are given a graph as input. undirected weighted graph.

$G=(V,E)$ and cost on edges costs are positive and we are given set of set T of terminal vertices $T \subseteq V$ and we need to compute a tree which contains T and of minimum edge weight. a minimum weight tree containing all vertices in T. The vertices in $V \setminus T$ are called Steiner vertices they are optional vertices it may be useful to use some Steiner vertices to connect the terminal vertices at lower cost. The vertices of V minus t are called stainer vertices ok.

So, what is the price collecting stainer tree? It is an extension or generalization of classical stainer tree. again the input is undirected weighted graph. $G=(V,E)$ cost on edges. and there are certain penalties. So, penalties associated with each vertex.

So, unlike the classical standard tree problem, we do not need to connect all the some set of terminal vertices. It is like if we whichever vertices we connect are part of tree for those vertices we do not incur a penalty and for other vertices which we omit for those vertices we pay the penalty. So, that is the penalty function $\pi : V \rightarrow R_{\geq 0}$ and this is the thing and a special this is the first special vertex called root $r \in V$. What is the output? compute a treaty which contains r and minimizes the sum of the cost of the edges that we are using in T, E[T] is the edge set of T that the costs plus the sum of the penalties of the vertices which do not belong to T.

V[T] is the vertex set of T is $\pi(v)$. So, why it is a generalization? So, in the classical

Steiner tree problem if we set the penalty of terminal vertices to be infinity and the penalty of Steiner vertices to be 0, then if we if we set in this price collecting Steiner tree problem penalty of some vertices to be infinity and the other vertices to be 0, then this is the standard tree problem because whichever vertex has penalty infinity that must belong to the output T and whichever vertices have penalty 0 they can be omitted without any penalty. So, the classical tenure tree problem is the same as the price collecting. standard tree problem where $\pi(i)$ or $\pi(v)$ is infinity if this v belongs to the set of terminals in the standard tree problem and 0 otherwise ok. So, it is a generalization.

So, now, because the standard tree problem even the classical version is NP complete this problem is also NP complete. Next, what we will do? We will write an integer linear programming formulation of this problem. We relax the integrality constraints in that formulation to get a linear programming relaxation of this problem and then we will use deterministic rounding. So, that is the high level idea. So, let us begin with writing an integer linear programming formulation of this problem.

So, we introduce we have a variable x_e for every edge e of this graph which will take value 1 if e is included in the output tree. We also have a variable for each vertex, variable say y_v for every vertex $v \in G$ which will take value 1 if v is included in the output tree and 0 otherwise. Here also if the edge e is not included in the output then x_e will take 0 otherwise. So, with this variables it is easy to write down the optimization function, we want to minimize the cost and plus the penalties of the vertices which are not part of the output tree.

So, minimize objective function. minimize the total cost of edges included. So, $e \in E$ $c_e x_e$. So, x_e will be 1 if this edge is included in the output otherwise it will be 0. So, this is the total cost of the edges included in the output tree plus for all vertex v.

So, let us call it say $i \in V$. Now if I do not include i in the output tree then I will incur a penalty of $\pi(i)$. So, this will be incurred only if y_i is 0. So, I will write $1 - y_i$. So, if y_i is 1 this penalty is not incurred.

So, this is the optimization function we want to minimize next what we need we need to write the constraints. So, that x_e and y_i are forced to take the values that we intend them to take. So, the first is r this root vertex must be in the tree. So, let me write constraints So, y_r must be equal to 1 and then what I want is whenever y_i is 1 for any i if y_i equal to 1 there is y_r I know y_r is always equal to 1. I want to ensure that there exist a path where all the edges I pick.

Now, how do I encode this? So, one way to encode is if I look at all subset of vertices

which contains r, but does not contain y and if I look at the edges with exactly one end point in S. at least one of the edges must be picked in my solution. So, for each $i \in S$ so, for each $i \in V$ if y_i equal to 1, then what I want is then I want for all subset $S \subseteq V \setminus \{i\}$ this S should include should not include i and it should include r. at least one of the edges must be picked summation $e \in \delta(S)$, $\delta(S)$ is the set of edges having exactly one end point. in S formally that is all the edges $e \in E[G]$ such that $E \cap S$ this cardinality is 1.

So, at least one of the edges must be picked. So, x_e this should be greater than equal to 1 if y_i equal to 1 if y_i equal to 0 we do not have any constraint. So, if I write this is greater than equal to y_i. it may not be clear why if I ensure that for all such r-i cuts at least one edge is picked there that should ensure why that there exist an r to i path. So, that needs a proof and so, let us prove it claim.

there exists an r to i path if and only if for all subset $S \subseteq V \setminus \{i\}$ $r \in S$ summation x_e at least 1 edge from $\delta(S)$ is picked proof. One direction is obvious that if there exist an r to i path that path must use at least one edge from $\delta(S)$ So, this direction So, at least 1 edge from every delta s must be picked. So, this is obvious. Since any r to i path must pick at least one edge must contain at least one edge from $\delta(S)$.

for every $S \subseteq V \setminus \{i\}$ and $r \in S$. The other direction is follows from max flow mean cut theorem, which says that the maximum flow from r to to is equal to the minimum capacity of one cut. for all $S \subseteq V \setminus \{i\}$ $r \in S$ at least one edge is picked from $\delta(S)$. then the size of minimum r-i cut is at least 1. one can send one unit of flow using the edges picked.

there exists an r to i path. So, not only this proves that these constraints are enough, it shows that although we have exponentially many constraints, this proof gives a polynomial time separation oracle. So, what is the final linear program? Linear programming relaxation minimize $\sum_{e \in E} c_e x_e + \sum_{i \in V} \pi(i)(1 - y_i)$ subject to summation what are the constraints that for all $S \subseteq V \setminus \{i\}$ and $r \in S$. at least one of the edge must be picked at least one of the x is must be 1. If y_i is 1 otherwise there is no constraint. So, greater than equal to y_i and y_r equal to 1.

and all this x is and y is they are binary valued either they should take either 0 or 1. So, y_i and x_e belongs to $\{0, 1\}$ this is for all $i \in V$ and $e \in E$. So, this is the ILP formulation to get the linear programming relaxation we relax the integrality constraints. We replace this integrality constraints with for all $i \in V, e \in E$ $0 \leq y_i \leq 1$ and $0 \leq x_e \leq 1$ ok. So, here we have exponentially many constraints, but you use this lemma this claim to show that to design a polynomial time separation oracle for this for this linear program.

So, that I give it as a homework. So, use max flow mean cut theorem to design a polynomial time separation oracle ok. So, hence we can use ellipsoid method to solve this linear program. Hence we can use the ellipsoid method to solve this LP in polynomial time ok. So, let us stop thank you.