18.433 Combinatorial Optimization

Separation Oracles

November 6,13 *Lecturer: Santosh Vempala*

In the last lecture, we presented a polynomial-time algorithm, namely the Ellipsoid algorithm, for solving a linear program. We also saw using binary search how optimization problems and feasibility problems are equivalent.

In the ellipsoid algorithm, we first consider an initial ellipsoid that contains our entire polyhedron P. If z, the center of the ellipsid, belongs to P then we can solve the feasibility problem and we are done. Otherwise we need to find a half-space $a_k x \leq b_k$ such that P lies inside of the half-space and z lies in the outside. Then we obtain another ellipsoid containing the intersection of our previous ellipsoid and the half-space. We iterate until we find a point inside the polyhedron or claim that there is no such point. Here the volume drops by a factor of $e^{\frac{-1}{2n+2}}$ in each iteration.

In today's lecture, we will see that the ellipsoid algorithm can be used in a much more general setting. The main thing we need is to be able to answer the question of whether z is in P or not and finding a separating hyperplane in the latter case. A procedure which does this is called a *separation oracle*.

Consider the *minimum-cost arborescence* problem: given a directed graph $G = (V, E)$, an special vertex $r \in V$ and a positive cost c_{ij} for each edge $(i, j) \in E$, find a subgraph of minimum cost that contains directed paths from r to all other vertices. The cost of the subgraph is the sum of the costs of its edges. This problem seems very similar to minimum spanning tree.

We solve this problem using linear programming. Let K = Convex Hull of $\{x^T \in R^{|E|} : T \textit{ is an arborescence}\}$ where $x_e^T = 1$ if $e \in T$ and $x_e^T = 0$ otherwise. Now our problem is $min\{c_{ij}x_{ij} : x \in K$. We can observe that this problem has the same optimal solution as the following problem:

$$\begin{aligned} & & min\ c^T x \\ \forall S \subseteq V \textit{ where } r \in S & \sum_{i \in S, j \notin S, (i,j) \in E} & x_{ij} \geq 1 \\ & & x_{ij} \in \{0, 1\} \end{aligned}$$

In fact, we can check that the above condition is equivalent to existence of paths from r to each other vertex $v \in V$. Furthermore, Edmonds proved that if we relax the last constraint to $0 \leq x_{ij} \leq 1$, then the set of feasible solutions to the linear program above is exactly K.

Now we have a linear program with an exponential number of constraints. However, we can design a separation oracle that runs in polynomial time. Checking $0 \leq x_{ij} \leq 1$ can be done easily in polynomial time. We can also test the first set of constraints by calling at most $(n-1)$ times of $min - cut$ procedure. We consider vertex r as the source, each vertex $s \in V, s \neq r$ as the sink and each x_{ij} as the capacity of edge (i, j) in the min-cut problem and check whether the minimum cut has capacity less than 1 or not. We can also find a violated constraint, namely a directed cut of value less than 1, if there exists such a cut.

Having the above facts, we can solve the minimum cost arborescence problem using ellipsoid algorithm in polynomial time.

In general, we have the following problem called *convex programming*: given a convex set K, find a point x in K. To solve this problem using ellipsoid algorithm, first we need a bounding ball for K. Then we need a separation oracle and finally find a lower bound on volume K (or the radius of a ball contained in K). It often turns out among these tasks, finding a separation oracle is the most difficult, since the other parts usually can be done very easily. For our previous example, the ball containing all points whose coordinates are between 0 and 1 is our initial ball. We observe that if the radius of our initial ball is R and the radius of our final ball is r, then the volume of the initial ellipsoid is $f(n)R^n$ and the volume of the final ellipsoid is greater than or equal to $f(n)r^n$ ($f(n)$ is a function of the dimension). Let i be the number of iterations. We must have

$$f(n)R^n e^{\frac{-i}{2n+2}} \geq f(n)r^n$$

and thus i is in $O(n^2 \log(R/r))$. In each iteration, we call the separation oracle once. Let its running time be $g(n)$.Then the overall running time is in $O(n^2 \log(R/r) \cdot g(n))$.

We now consider another problem for which the above approach can be applied. The *maximum independent set* problem is defined as follows: given a graph $G = (V, E)$, find a subset $S \subseteq V$ of maximum size such that there is no edge between vertices of S. The integer program (IP) for this problem is:

$$\begin{aligned} & max \sum_i x_i \\ \forall (i,j) \in E \quad & x_i + x_j \leq 1 \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

and its relaxed linear program (LP) is:

$$\begin{aligned} & max \sum_i x_i \\ \forall (i,j) \in E \quad & x_i + x_j \leq 1 \\ & 0 \leq x_{ij} \leq 1 \end{aligned}$$

However $Opt(LP)/Opt(IP)$ can be large, e.g. for a complete graph $Opt(LP) = n/2$ by setting all x_i's equal to $1/2$ but $Opt(IP) = 1$, and thus solving the linear program does not even give a good approximation of the optimum integer solution.

Let us add another set of constraints to the linear program. We can observe that for each odd cycle C, we can choose at most $\frac{|C|-1}{2}$ vertices in an independent set. Thus our new LP is as follows:

$$\begin{aligned} & max \sum_i x_i \\ \forall (i,j) \in E \; & x_i + x_j \leq 1 \\ \forall \text{ odd cycles } C \; & \sum_{i \in C} x_i \leq \frac{|C|-1}{2} \\ & 0 \leq x_{ij} \leq 1 \end{aligned}$$

We note that after adding this set of constraints, the linear program and the integer program are still different (the counter-example is left for an exercise).

Let us find a separation oracle for these constraints. The first and the third sets can be checked in polynomial time. To test the second set, we define $y_{ij} = 1 - x_i - x_j$ for each edge $(i,j) \in E$. Now we can observe that the second set of constraints is equivalent to saying that for any odd cycle C, $\sum_{(i,j) \in E(C)} y_{ij} \geq 1$. So by finding the length of the minimum odd cycle in the graph G, we can give a separation oracle.

We show that a minimum length odd-cycle can be found in polynomial time. To this end, we construct an undirected bipartite graph $G' = (V' = V_1 \cup V_2, E')$ such that there exist a vertex $i_1 \in V_1$ and a vertex $i_2 \in V_2$ corresponding to each vertex $i \in V$. For an edge $(i,j) \in E$, we add two edges (i_1, j_2) and (j_1, i_2) in E'. We see that odd cycles in G correspond to paths from i_1 to i_2 in G' and finding a shortest path from i_1 to i_2 is easy to perform in polynomial time. Thus we have a polynomial-time separation oracle.