600.469 / 600.669 Approximation Algorithms **Lecturer:** Michael Dinitz
Topic: Linear Programming **Date:** 2/24/15
Scribe: Runze Tang

9.1 Linear Programming

Suppose we are trying to approximate a minimization problem (everything will hold for maximization problems too, but with the inequalities reversed). A general way to prove α-approximation is as following:

1. Prove OPT $\geq$ LB
2. Prove ALG $\leq \alpha\cdot$ LB

Then we can conclude ALG $\leq \alpha\cdot$ OPT. This is how essentially all of our analyses have worked so far.

9.2 Integer Linear Program

- A set $\{x_1, x_2, \cdots, x_n\}$ of variables, each of which must be an integer
- m linear inequalities over variables
- Possibly a linear objective function over variables

9.3 LP Rounding

We want to "round" the solution to LP to get a feasible solution to ILP. The general steps are as following:

1. Write ILP
2. Relax to LP (i.e. remove the integrality constraints)
3. Solve LP, getting solution $\vec{x}^*$
4. "Round" $\vec{x}^*$ to integer values to get a solution to ILP

Key ideas of rounding:

1. LP $\leq$ OPT, since it is a relaxation. Slightly more formally, any integral solution is obviously a solution to the LP, and hence the optimal LP value is at most the optimal ILP value, which equals OPT.

2. We design our rounding in order to guarantee that the integral solution we get back is at most $\alpha \cdot$ LP.

Putting these together, we get that ALG $\leq \alpha$ OPT. And instead of having to find some mysterious lower bound LB to compare the algorithm to, we can compare the output of the algorithm to the fractional LP solution, and only analyze how much our rounding increased the cost.

9.4 Weighted Vertex Cover

9.4.1 Problem Description

- **Input**: a graph $G = (V, E)$, and a cost function $c : V \to \mathbb{R}^+$
- **Feasible solution:** $S \subseteq V$ such that $\forall \{u, v\} \in E$, either $u \in S$ or $v \in S$
- **Objective:** $\min \sum_{v \in S} c(v)$, i.e. $\min c(S)$

9.4.2 Equivalent Integer Linear Program

$$\begin{array}{rrll} \text{minimize:} & \displaystyle\sum_{v \in V} c(v) \cdot x_v & & \\ \text{subject to:} & x_u + x_v \geq & 1 & \text{for each edge } \{u, v\} \in E \\ & x_v \in & \{0, 1\} & \text{for each vertex } v \in V \end{array}$$

It is easy to see that this ILP is an exact formulation of the weighted vertex cover problem. For one direction, suppose that we are given a vertex cover S. Then setting $x_v = 1$ if $v \in S$ and $x_v = 0$ if $v \notin S$ gives a solution to the ILP with cost exactly $c(S)$. For the other direction, suppose that we are given a solution $\vec{x}$ to the ILP. Then let $S = \{v \in V : x_v = 1\}$. Then S is a vertex cover with cost exactly equal to the objective value of the ILP on $\vec{x}$. Hence the ILP and weighted vertex cover are equivalent.

9.4.3 Relax to a Linear Program

Now we would like to drop integrality constraints to get LP:

$$\begin{array}{rrll} \text{minimize:} & \displaystyle\sum_{v \in V} c(v) \cdot x_v & & \\ \text{subject to:} & x_u + x_v \geq & 1 & \text{for each edge } \{u, v\} \in E \\ & 0 \leq x_v \leq 1 & & \text{for each vertex } v \in V \end{array}$$

Theorem 9.4.1 *If $\vec{x}$ is feasible for ILP, then $\vec{x}$ is feasible for LP.*

Proof: $\vec{x}$ satisfies the edge constraints of the LP since it satisfies them for the ILP, and $x_v \in \{0, 1\}$ for each $v \in V$ due to the integrality constraints of the ILP, and hence $0 \leq x_v \leq 1$. ■

Corollary 9.4.2 *OPT(LP) $\leq$ OPT(ILP).*

Proof: By the theorem above, the optimal solution to ILP is feasible to LP. ■

Corollary 9.4.3 *If we find a solution of cost less or equal to $\alpha\cdot$ OPT(LP), then we have an α-approximation algorithm.*

Proof: cost(SOL) $\leq \alpha\cdot$ OPT(LP) $\leq \alpha\cdot$ OPT(ILP) ■

9.4.4 LP Rounding

As always, we first solve the LP to get an optimal fractional solution $\vec{x^*}$. We can do this in polynomial time since the LP has size polynomial in the size of the instance. We then round $\vec{x^*}$ to an integral solution $\vec{x'}$ as follows: for each $v \in V$, we set $x'_v = 1$ if $x^*_v \geq \frac{1}{2}$, and set $x'_v = 0$ otherwise.

Theorem 9.4.4 *$\vec{x'}$ is a feasible solution to ILP.*

Proof: For any $(u, v) \in E$, we have $x^*_u + x^*_v \geq 1$, which implies that $\max(x^*_u, x^*_v) \geq \frac{1}{2}$. So either $x'_u = 1$ or $x'_v = 1$. Thus $x'_u + x'_v \geq 1$. ■

Theorem 9.4.5 *$c(\vec{x}') \leq 2c(\vec{x}^*)$.*

Proof: $c(\vec{x}') = \sum_{v\in V} c(v)\cdot x'_v = \sum_{\{v\in V: x^*_v \geq 1/2\}} c(v) \leq \sum_{v\in V} c(v)\cdot 2x^*_v = 2c(\vec{x^*})$. ■

Corollary 9.4.6 *This rounding is a 2-approximation algorithm.*

9.5 Weighted Set Cover

9.5.1 Problem Description

Weighted set cover problem is the set cover problem with costs on each set.

9.5.2 Equivalent Integer Linear Program

$$\begin{array}{rrll}
\text{minimize:} & \displaystyle\sum_{S\in\mathcal{S}} c(S)\cdot x_S & & \\
\text{subject to:} & \displaystyle\sum_{\{S: e\in S\}} x_S \geq & 1 & \text{for each element } e \in U \\
 & x_S \in & \{0,1\} & \text{for each set } S \in \mathcal{S}
\end{array}$$

This is clearly an exact formulation – it is easy to see that solutions to the ILP correspond to set covers with the same cost, and vice versa.

9.5.3 Relax to a Linear Program

$$\begin{aligned} *\text{minimize:} \quad & \sum_{S \in \mathcal{S}} c(S) \cdot x_S \\ \text{subject to:} \quad & \sum_{\{S : e \in S\}} x_S \geq \ 1 \quad \text{for each element } e \in U \\ & 0 \leq x_S \leq 1 \qquad \text{for each set } S \in \mathcal{S} \end{aligned}$$

9.5.4 LP Rounding

In a previous lecture on set cover we defined the *frequency* of an element e to be $f_e = |\{S \in \mathcal{S} : e \in S\}|$ and defined $f = \max_{e \in U} f_e$. Given $\vec{x}^*$ which is solution to LP, set $x'_S = 1$ if $x^*_S \geq \frac{1}{f}$, and set $x'_S = 0$ otherwise.

Theorem 9.5.1 *$\vec{x}'$ is a feasible solution to ILP.*

Proof: Since each elements is in at most f sets, the LP constraint $\sum_{\{S:e\in S\}} x^*_S \geq 1$ implies that $\max_{\{S:e\in S\}} x^*_S \geq \frac{1}{f}$. Thus there exists at least one S which contains e such that $x'_S = 1$. So $\sum_{\{S:e\in S\}} x'_S \geq 1$. ■

Corollary 9.5.2 *It is an f-approximation algorithm.*

Proof: By the previous theorem $\vec{x'}$ is a feasible set cover. For its cost, we have that $c(\vec{x'}) = \sum_{S\in\mathcal{S}} c(S)x'_S \leq f \cdot \sum_{S\in\mathcal{S}} c(S)x^*_S = f \cdot c(\vec{x^*})$. ■

9.6 Integrality Gap

Definition 9.6.1 *The integrality gap of an LP is* $\sup_{instance\ I} \left(\frac{OPT(I)}{LP(I)}\right)$.

The integrality gap of an LP measures the strength of the relaxation. If we prove that $ALG \leq \alpha \cdot OPT$ by proving that $ALG \leq \alpha \cdot LP$ (as we have been doing with rounding), then we cannot give a ratio α that is better than the integrality gap (or else on the instance I which achieves the integrality gap we would be able to round the LP solution to a value less than OPT, giving a contradiction).

9.6.1 Integrality Gap for Weighted Vertex Cover

Consider the instance to be: $G = K_n$, which is the complete graph, and $c(v) = 1$ for all $v \in V$.

Theorem 9.6.2 *The integrality gap is at least* $2\left(1 - \frac{1}{n}\right)$.

Proof: OPT $= n - 1$, since if there are two nodes not in the vertex cover the edge between them will not be covered. But in the LP, if we set set $x_v = \frac{1}{2}$ for every $v \in V$ we get a valid LP solution with cost $\frac{n}{2}$. Hence $IG \geq \frac{n-1}{n/2} = 2\left(1 - \frac{1}{n}\right)$. ■

9.6.2 Integrality Gap for Max Independent Set

The ILP is:

$$\begin{array}{rrll} \text{maximize:} & \displaystyle\sum_{v \in V} x_v & & \\ \text{subject to:} & x_u + x_v \leq & 1 & \text{for each edge } \{u,v\} \in E \\ & x_v \in & \{0,1\} & \text{for each vertex } v \in V \end{array}$$

Theorem 9.6.3 *The integrality gap is at least $\frac{n}{2}$.*

Proof: Consider the instance to be $G = K_n$, which is a complete graph. So OPT $= 1$ since every pair of vertices are connected. For the LP, set $x_v = \frac{1}{2}$ for every $v \in V$. Then we get the LP cost to be $\frac{n}{2}$. Hence the integrality gap is at least $\frac{n/2}{1} = \frac{n}{2}$. ∎

9.7 Solving LPs

This is a bit outside the scope of this class, but we will talk a little bit about algorithms for solving LPs. This section is pretty informal, but everything can be made formal.

9.7.1 Simplex

Note that the feasible region of an LP is a polytope (by definition) and hence is convex (for every two points in the feasible region, their midpoint is also feasible). The oldest heuristic for solving LPs is the simplex algorithm. We won't talk much about this algorithm, but the high-level view is straightforward. Given a polytope, there is a natural graph associated with it where the vertices of the graph are the vertices of the polytope (points which are tight for d linearly independent constraints of the polytope, where d is the dimension) and two vertices are adjacent if in the polytope they are also adjacent (the line segment between them is tight for $d-1$ linearly independent constraints). This algorithm starts by finding a vertex of the polytope, and then moving to a neighbor with decreased cost as long as this is possible. By linearity and convexity, once it gets stuck it has found the optimal solution.

Unfortunately simplex does not run in polynomial time – even with a polynomially-sized LP, the number of vertices of the polytope can be exponential. Simplex does well in practice, but poorly in theory.

9.7.2 Interior Point Methods

While simplex only moves along the outer faces of the polytope, there are algorithms known as "interior-point"methods which make moves *inside* the polytope. There are now known to be interior-point methods which have polynomial running time, and are also extremely fast in practice. So in both theory and in practice, we can solve LPs efficiently.

9.7.3 Ellipsoid

We will spend a bit more time talking about an algorithm known as *Ellipsoid*, which works well in theory but poorly in practice. Nevertheless, it has some properties which are extremely nice theoretically, so we will feel free to use it when designing algorithms.

As a first step, we will reduce optimization to feasibility. Suppose that we are given an algorithm which can determine whether a given LP is feasible, i.e. it returns YES if the feasible region is non-empty and NO otherwise. Now suppose that we are given an LP which we want to solve. For any value T, we can use our feasibility algorithm to see if there is an LP solution with cost at most T by adding a single new constraint (the objective is at most T). So we can do a binary search over possible values of T to find in polynomial time the smallest T for which the feasibility region is nonempty. Hence if we have such a feasibility algorithm, we can also solve LPs with objective functions.

9.7.3.1 The algorithm

The ellipsoid algorithm is an algorithm for testing feasibility. Informally, it is the following.

Algorithm 1 Using Ellipsoid to Check Feasibility Algorithm

```
Input: Convex constraints
Output: A feasible solution if one exists.
  Find an ellipsoid containing the polytope.
  Let $\vec{x}$ be the the center of the ellipsoid.
  $k = 0$
  while $\vec{x}$ is not feasible AND $k <$ maxiter do
    $k = k + 1$
    Find a constraint violated by $\vec{x}$
    Draw a parallel hyperplane corresponding to this violated constraint through $\vec{x}$. This divides
    the ellipsoid into two parts with equal volume, one of which is on the "wrong" side of the
    hyperplane and so is entirely infeasible.
    Consider the part which is on the correct side of the violated constraint (i.e. the part in which
    the polytope must appear if it is nonempty). Find a new ellipsoid containing this part.
    Let $\vec{x}$ be the center of the new ellipsoid.
  end while
  return $\vec{x}$, which is a feasible solution; or no feasible solution exists.
```

It is not hard to see that this algorithm is correct, in that if the polytope is nonempty it will eventually find a feasible point and if the polytope is empty then it will never find a feasible point. The hard part is proving that it takes only polynomial time in the worst case, i.e. that we can set maxiter to a polynomial value. While it is complicated, the key idea is that due to the geometry of ellipsoids, in each iteration the volume of the ellipsoid that we are considering decreases by a constant factor.

9.7.3.2 Separation

The ellipsoid algorithm has an extremely nice property: in order to make it work, all that we need to be able to do is find a violated constraint given some point $\vec{x}$ (or prove that $\vec{x}$ is feasible). The is is called the *separation* problem. If the LP is small then this is simple – we can just check all of the constraint. But sometimes our LPs are not small, and ellipsoid still lets us solve them! If there are an exponential number of constraints, we cannot even write down all of them in polynomial time so we certainly cannot use simplex or interior-point methods to solve the LP. But if we can separate, then ellipsoid lets us solve in polynomial time despite not even being able to write down the LP. Let's do a somewhat trivial example of this: the spanning tree polytope.

9.7.4 Spanning Tree Polytope

Consider the minimum spanning tree problem. Obviously we know multiple simple and fast algorithms. But suppose we want to do this by an LP. While it's not immediately obvious how to write such an LP, it turns out that the following formulation is good:

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c(e) \cdot x_e & & (9.7.1)\\
\text{subject to:} \quad & \sum_{\{e \in E(S,\bar{S})\}} x_e \geq 1 & \text{for each cut } S \subseteq V \quad & (9.7.2)\\
& 0 \leq x_e \leq 1 & \text{for each } e \in E \quad & (9.7.3)
\end{aligned}$$

Note that the tree constraints are hidden in the optimization problem above since we can always make the objective function smaller by throwing away an edge.

Although it has exponentially many constrains, we can separate in polynomial time. Given $\vec{x}$, first run the min-cut algorithm. If the min cut is larger or equal to 1, than $\vec{x}$ is a feasible solution; otherwise the minimum cut S corresponds to a violated constraint. Hence we can separate, so by the ellipsoid algorithm we can optimize.