CMPUT 675: Approximation Algorithms **Fall 2015**

Lecture 3 (Sep 8, 2015): Intro to LP

Lecturer: Mohammad R. Salavatipour *Scribe: Nadia Ady*

3.1 Recall: Set Cover

During last week's lecture, we introduced the Set Cover problem:

input:	ground set	$U = \{e_1, ..., e_n\}$,
	collection of sets	$S = \{S_1, ..., S_m\}$ s.t. $\forall 0 \le i \le m, S_i \subseteq U$, and
	cost function	$c : S \to \mathbb{Q}^+$
goal:	find a minimum-cost collection of S,	S' s.t. $\bigcup_{S_i \in S'} S_i = U$.

We discussed the natural greedy algorithm and showed that it is an H_n-approximation algorithm.

3.1.1 A tight example for Greedy-SC

Today, we can show that this analysis of the natural greedy algorithm as an H_n-approximation is tight with a simple example. Suppose our input collection of sets S consists of, firstly, for each element e_i of the ground set U, the singleton set containing just that element, $\{e_i\}$, and secondly, the ground set itself, U. Further, suppose the cost function c is as follows:

$$\begin{array}{rccccc} S = & \{\{e_1\}, & \{e_2\}, & \cdots, & \{e_n\}, & U\} \\ & \downarrow & \downarrow & & \downarrow & \downarrow \\ c: & \frac{1}{n} & \frac{1}{n-1} & \cdots & 1 & 1+\varepsilon \end{array}$$

We see that, initially, the cost effectiveness for any singleton set $\{e_i\}$ is $\frac{1}{n(n+1-i)}$, and in particular, in the first iteration, the cost per element covered for $\{e_1\}$ is $\frac{1}{n} < \frac{1+\varepsilon}{n}$, so the greedy algorithm will pick $\{e_1\}$. In fact, in any iteration, say when choosing the ith set to add to T, the cost effectiveness of $\{e_i\}$ will be $\frac{1}{n+1-i}$ while the cost effectiveness of U will be $\frac{1+\varepsilon}{n+1-i}$. This means that U will never be chosen before any of the, and so T will consist of $S - U$.

For this example, the cost comparison is:

$$\text{OPT} = 1 + \varepsilon$$

$$\text{greedy} = \frac{1}{n} + \frac{1}{n-1} + \cdots + 1 = H_n \approx \ln n$$

As ε may be chosen arbitrarily close to 0, we see that the natural greedy algorithm has produced a solution H_n times the optimal.

3.2 Linear Programming in the design of Approximation Algorithms

Linear Programming (LP) is the problem of optimizing a linear function of variables subject to a set of linear constraints.

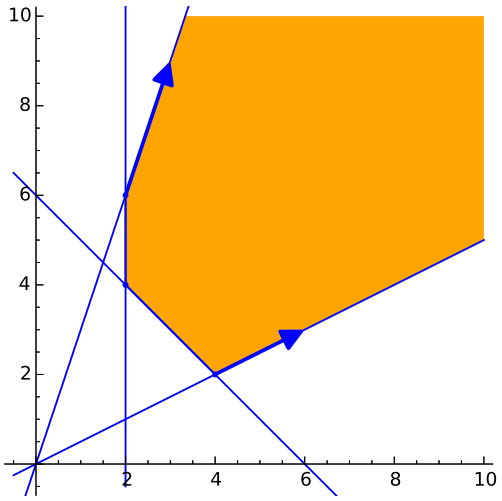


Figure 3.1: This figure depicts the polyhedron described by the constraints in example 3.2.

Definition 1 *A linear programming problem takes the standard form:*

$$\textit{determine } n \textit{ variables: } x_1, ..., x_n \in \mathbb{R}, \textit{ with } x_j \geq 0 \textit{ for } 0 \leq j \leq n \tag{3.1}$$

$$\textit{minimize: } \sum_{j=1}^{n} c_j x_j \tag{3.2}$$

$$\textit{subject to: } \sum_{j=1}^{n} a_{ij} x_j \geq b_i \tag{3.3}$$

We call the number of variables, n, the dimension *of the LP, and say m is the number of constraints. A* feasible solution *to an LP problem is any assignment to the variables that satisfy all of the constraints.*

In matrix form, an LP looks like

$$\begin{array}{rrcll} \text{minimize}: & x \cdot c & & & \\ \text{subject to}: & x \cdot A & \geq & b, & \forall 1 \leq j \leq m \\ & x & \geq & 0, & \forall 1 \leq i \leq n \end{array}$$

where x is an n-dimensional vector of variables. In such a case, A is called the *constraint matrix.*

Example 1 *Suppose we have the variables x_1, x_2 and we simply want to minimize x_2 (i.e. we want to minimize $0x_1 + 1x_2$) subject to the following constraints.*

$$\begin{aligned} x_1 &\geq 2 \\ 3x_1 - x_2 &\geq 0 \\ x_1 + x_2 &\geq 6 \\ -x_1 + 2x_2 &\geq 0 \end{aligned}$$

Each LP falls into exactly one of the following three categories:

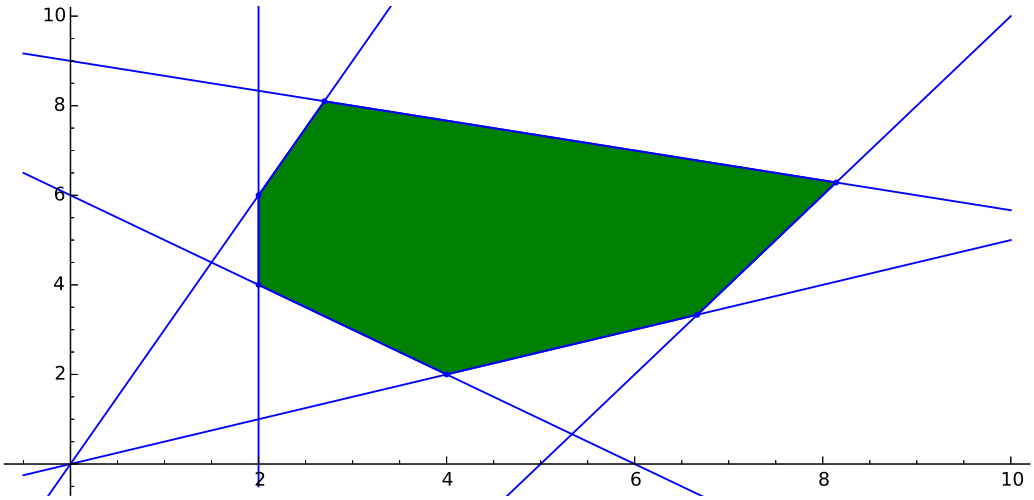


Figure 3.2: This is a visualization of an example polytope.

- There is a feasible $x \in \mathbb{R}^n$ such that for all feasible $x' \in \mathbb{R}^n$, the objective function value under assignment x' is at least the objective function value under x. We say the LP has a *finite optimum*.
- Every $x \in \mathbb{R}^n$ is not feasible. In this case, we say the LP is *infeasible*.
- For every $z \in \mathbb{R}$, there exists a feasible $x \in \mathbb{R}^n$ such that the objective function value under assignment z exceeds the objective function value under assignment x. Here, the LP is said to be *unbounded*.

Each constraint defines a half-space of $\mathbb{R}^n$. The feasible region is the intersection of all half-spaces defined by the constraints of the LP. Consider an LP with a non-empty feasible region. If this feasible region is bounded then the LP has a finite optimum which also implies that the feasible region is infinite if the LP is unbounded.

Definition 2 *A subset $P \subseteq \mathbb{R}^n$ is called* convex *if for every pair $x, x' \in P$, and any $0 \leq \alpha \leq 1$, $x^{\sim} = \alpha x + (1-\alpha)x' \in P$ as well.*

We call the set of feasible solutions a *polyhedron*; it is always convex. Intuitively, if you take any two points in the feasible region and draw a straight line between them, all points on that line are also feasible, so the region is convex. If the feasible region is bounded, then we have a *polytope*.

A vertex/extreme point/basic feasible solution is a point that is an intersection of n constraints. Notice that a basic feasible solution (bfs) cannot be written as a convex combination of other feasible solutions.

Theorem 1 *If an LP has a finite optimum, then it has an optimum solution that is a bfs.*

In the computational version, each number must be represented as a rational number, $\frac{p}{q} \in \mathbb{Q}$, so we need $\Delta = \log p + \log q$ bits to represent $\frac{p}{q}$.

Theorem 2 *LP can be solved in time poly(n, m, Δ).*

A linear program can be solved in polynomial time using, for example, the ellipsoid method or an interior point method. First we note that the difficulty of linear programming is not in finding an optimum solution, it is

rather finding a feasible solution. In other words, it is an easy exercise to show that having access to a procedure that finds a feasible solution one can optimize over the feasible region. So the main task is to find a feasible soltuion. The ellipsoid method works in the following way. We start with a large ellipsoid that is containing the convex set P (feasible region); the question of how to start with such a ellipsoid is something we skip at this point. Then we check if the center of this ellipsoid is inside P or not. If it is then we have found a feasible solution and we are done. Otherwise, the algorithm finds a linear inequality that is satisfied by all the points in the feasible region and is not satisfied by the center point. Then it tries to find a smaller ellipsoid that contains all the region of the original ellipsoid that is to one side of that linear constraint (separating the center point from P). This is guaranteed to contain P and is guaranteed to have a smaller volume by a factor. One significant feature of this method is that one can solve even LP's with exponentially manin constraints as long as there is a polynomial time separation oracle. An *oracle* for an LP is a method that decides if a proposed point $x \in \mathbb{R}^n$ is feasible and, if not, produces a violated constraint. The ellipsoid method, while not nearly as practical as interior point methods, has the advantage of solving an LP in polynomial time if there is a polynomial time oracle for the problem even if the number of constraints is exponential. We will use this property in the design of approximation algorithms.

3.2.1 Vertex Cover

We recall the Vertex Cover problem: given a graph, $G(V, E)$, and a cost function, $c : V \to \mathbb{Q}^+$, find a minimum cost vertex cover. We can translate this problem into a set of variables and linear constraints (an LP) as follows:

Our set of variables is $\{x_u \in \{0, 1\} | u \in V\}$. In this way, we have a binary variable assigned to each vertex : $x_u = 1$ if u is picked for the vertex cover and $x_u = 0$ if not. The objective function is to minimize $\sum_{u \in V} c(u)x_u$ subject to the constraint that for each edge $uv \in E(G)$, $x_u + x_v \geq 1$. This constraint corresponds to ensuring that the assignment defines a cover (at least one vertex on each edge is picked for the vertex cover). To convert this Integer Programming (IP) problem into an LP, we simply drop the requirement that $x_u \in \{0, 1\}$ for each vertex u.

$$\begin{aligned} \text{minimize:} \quad & \sum_{i=1}^{n} c(u) \cdot x_u & & & \\ \text{s.t} \quad & x_u + x_v & \geq & 1, & \forall \ uv \in E \\ & x_u & \geq & 0 & \end{aligned}$$

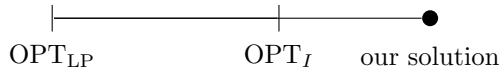
3.2.2 VC using LP rounding

VC-LP-ROUNDING

```
1  let x* be an optimum LP solution
2  for each vertex u ∈ V(G)
3        do if x*_u ≥ 1/2
4                 then x̂_u ← 1
5              else x̂_u ← 0
6  return x̂
```

Lemma 1 *The algorithm* VC-LP-ROUNDING *returns a vertex cover.*

Proof. For every edge uv, since $x_u + x_v \geq 1$ we can't have both $x_u, x_v < \frac{1}{2}$, so at least one of them is rounded up, and so at least one of u, v is chosen for the vertex cover. ■



Lemma 2 *The algorithm* VC-LP-ROUNDING *is a 2-approximation algorithm.*

Proof. Let OPT_{LP} be the optimum value of the LP and OPT_{int} be the optimum integer solution. Clearly $\text{OPT}_{\text{LP}} \leq \text{OPT}_{\text{int}}$. Also since for every vertex u that we pay the full cost $c(u)$ optimum LP pays at least $\frac{1}{2}c(u)$ (because it must have been $x_u \geq 1/2$, therefore the cost of our solution is atmost $2 \cdot \text{OPT}_{\text{LP}}$. ■

There are often multiple ways to write an LP and they might have different integrality gaps (described below) We usually try to find a "good" LP, i.e. one that can be used to derive an integer solution that is not far from optimum LP.

3.3 The Integrality Gap

For an instance I of a minimization problem, let $OPT_f(I)$ be the cost of the optimal LP solution. Then, integrality gap of this instance is given by $OPT(I)/OPT_f(I)$. The integrality gap of the problem Π is $\max_I OPT(I)/OPT_f(I)$, i.e. the largest ratio between the fractional and integral solution over all possible instances of Π. Note that in any LP rounding algorithm, we use the LP solution as a lower bound for the optimal solution. Therefore, any such algorithm cannot have a performance ratio better than the integrality gap. In general, it is difficult to design algorithms with ratio better than the integrality gap. Therefore, large integrality gaps typically are indications of difficulty of a problem. Consider for example the Vertex Cover problem, and let I be the complete graph K_n. Clearly by assigning a value of $\frac{1}{2}$ to each vertex we get a fractional solution of value $\frac{n}{2}$ whereas any integer solution must contain at least $n-1$ nodes (or else one edge is not covered). Thus the integrality gap of V.C. is at least $2 - \frac{2}{n}$.

Exercise. Prove that the integrality gap for Set Cover problem is $\Omega(\log n)$.

Our α for any α-approximation using LP cannot be better than the associated integrality gap.

3.4 Set Cover Using LP

There are at least three different algorithms using LP to approximate Set Cover; today we will look at two of them.

3.4.1 IP/LP formulation for Set Cover

For every set $s \in S$, we have a variable x_s $in\{0,1\}$. However, to formulate this for LP, we relax this so that $0 \leq x_s \leq 1$. Note that the requirement that $x_s \leq 1$ is redundant as this must be true in any optimum solution. We aim to minimize $\sum c_s \cdot x_s$ subject to the constraint that for every element $e \in U$, $\sum_{s \ni e} x_s \geq 1$.

$$\begin{aligned} \text{minimize}: \quad & \sum_{i=1}^{m} c(S_i) x_{S_i} \\ \text{subject to}: \quad & \sum_{e \in S_i} c(S_i) x_{S_i} \geq 1, \quad \forall e \in U \\ & x_{S_i} \geq 0, \end{aligned}$$

For convenience, we define the *frequency* of an element e as the number of sets that contain e, and denote the maximum frequency over all elements f.

The following outlines a simple algorithm using deterministic rounding for the Set Cover problem.

1. Let x^* be an optimum LP solution.
2. $\forall$ set $s \in S$, if $x^*_s \geq \frac{1}{f}$, then $\hat{x}_s \leftarrow 1$; otherwise $\hat{x}_s \leftarrow 0$.
3. return $\hat{x}$

Lemma 3 *This returns a feasible set cover solution.*

Proof. Let $I = \left\{j \mid S_j \text{ has } \hat{x}_{S_j}\right\}$. Assume that $e \notin \bigcup_{j \in I} S_j$. Then for all sets S_j such that $e \in S_j$, we must have $x^*_{S_j} < \frac{1}{f}$ (else $\hat{x}_{S_j}$ would have been rounded up). This implies that $\sum_{S_j \ni e} x^*_{S_j} < \sum_{S_j \ni e} \frac{1}{f} = 1$, which values the constraint for e. ■

3.4.2 Set Cover Using Randomized Rounding

Now we use randomized rounding algorithm to solve Set Cover. The intuitive idea behind this algorithm is that we will treat x^* as a probability distribution and then round each of them up with probablity equal to the value given by the LP solution.

1. Start with an empty collection of sets $C \leftarrow \varnothing$.
2. Let x^* be an optimum LP solution.
3. For each set S_i, set $\hat{x}_{S_i} = 1$ with probability $x^*_{S_i}$.
4. Let $C = \{S_i | \hat{x}_{S_i}\}$
5. Return C

First, let's compute the expected value of the cost of the collection C.

$$\begin{aligned} E[\text{cost of } (C)] &= \sum_{s_j} \Pr[S_i \text{ picked}] \cdot c(s_j) \\ &= \sum_{s_j} c(s_j) \cdot x^*_{s_j} \\ &= \text{OPT}_{\text{LP}} \end{aligned}$$

But the problem with this set is that the solution might not be a set cover by itself. To address this, we repeat this procedure $\alpha \ln n$ times; we will show that with sufficiently high probability the returned solutionis a good set cover. That is, if C_i is the collection of sets that are selected at the i^{th} iteration of this algorithm, then we would have the final answer as $\mathcal{C} = \cup_{i=1}^{\alpha} C_i$.

Consider an arbitrary element e_j and WLOG assume that it belongs to sets $S_1, \ldots, S_k$. Since we start from a feasible solution, we must have $x^*_{S_1} + \ldots + x^*_{S_k} \geq 1$; in the worst case we should have $x^*_{S_1} + x^*_{S_2} + \ldots + x^*_{S_k} = 1$. What is the probability that none of the corresponding k sets be selected given the probabilities of $x^*_{S_i}$'s? The following lemma (whose proof is starightforward) suggests that this probability is the highest when all the x^*'s are equal.

Lemma 4 *Given $x_1 + \ldots + x_k = 1$, if we select each S_i with probability x_i^*, the probability that no S_i will be selected is highest (worst case) when all $x_i^* = \frac{1}{k}$.*

Thus the probabilty that an element e_j is not covered in the i'th iteration of this algorithm is:

$$\Pr\left[e_j \notin C_i\right] \leq \left(1 - \frac{1}{k}\right)^k \leq e^{-1} \tag{3.4}$$

Now one way to increase the probability to get a set cover is to repeat the algorithm $\alpha \ln n$ times and take union of all the C's i.e. sets selected in each iteration; this is exactly what we do. Thus the probablity of an element not covered after $\alpha \ln n$ iteration is

$$\Pr\left(e_j \notin \bigcup_{i=1}^{\alpha \ln n} C_i\right) \leq e^{-\alpha \ln n} \tag{3.5}$$

$$= \frac{1}{4n} \tag{3.6}$$

Using union bound, the probability that the result after $\alpha \ln n$ iterations is not a set cover is at most

$$\Pr\left(\bigcup_{i=1}^{\alpha} C_i \text{ is not a set cover}\right) \leq \frac{n}{4n} \tag{3.7}$$

$$= 1/4. \tag{3.8}$$

The expected cost for the probablistic rounding set cover after $\alpha \ln n$ iterations is

$$E\left[cost\left(\bigcup_{i=1}^{\alpha} C_i\right)\right] = \alpha \ln n \cdot \mathrm{OPT}_{\mathrm{LP}} \tag{3.9}$$

$$= O\left(\log n \cdot \mathrm{OPT}\right) \tag{3.10}$$

Using Markov's inequality, $\Pr[X \geq t] \leq \frac{E[X]}{t}$:

$$\Pr\left[cost(C) > 4\alpha \cdot \mathrm{OPT}_{\mathrm{LP}}\right] \leq \frac{1}{4}$$

Therefore, with a probability greater than or equal to $\frac{1}{2}$, we have a feasible solution with cost less than or equal to $4\alpha \cdot \mathrm{OPT}_{\mathrm{LP}}$ which is in $O(\log n) \cdot OPT$, where OPT is the cost of the optimal solution. To get better probability it's enough to repeat the algorithm a constant number of times and get the best answer.