CSE525: Randomized Algorithms and Probabilistic Analysis April 11,2013

Lecture 4

Lecturer: Anna Karlin *Scribe: Travis Mandel, Daniel Yang Li*

1 Introduction

One useful application of random algorithms is to approximate NP-hard problems: although there is no known polynomial time algorithm for finding the optimal solution these problems, it is often possible to find a randomized algorithm that can find an approximately optimal solution. One common method for doing this is called **Randomized Rounding of Linear Programs(LPs)**. The basic idea is to take an NP-hard problem, relax the constraints such that it becomes a linear program, which we know how to solve efficiently, and then somehow convert that solution back to an approximate solution to the original problem. The basic recipe is as follows:

1. Formulate problem as an Integer Program(IP), where all variables must be either 0 or 1. In general, finding an optimal solution to an Integer Program is NP-complete.
2. Relax the Integer Program to a linear program where all variables are real values in the interval $[0, 1]$. This is called "relaxing" because every solution to an integer program is also a solution to a linear program, but it doesn't go the other way: the linear solution is unlikely to be a solution to the original integer program.
3. Solve the LP.
4. Construct a solution to the original problem by randomly rounding the result of the LP.
5. Analysis of the algorithm. Bound the quality of the randomly rounded by comparing it to the LP solution.

We will proceed by examining a few examples.

2 Example 1: Weighted Set Cover

2.1 Problem Statement

The set cover problem is an NP-complete problem defined as follows:

Imagine you have a set of elements $E = \{e_i, \ldots, e_n\}$, and a set of m subsets of E, $S_1, \ldots, S_m$ with weights $w_1, \ldots, w_m$. The goal is to find a set $I \subseteq \{S_i, \ldots, S_m\}$ such that:

- all elements are covered by I, and
- the sum of the weights of the subsets in I is minimized.

(You may have heard of this problem before in the unweighted case, where all the weights are one, in which case the goal is to minimize the number of subsets used.)

Since the problem is NP-complete, we are not looking for the optimal solution but rather an approximate solution. To compare approximation algorithms to each other, we define the term α-approximation. An α-approximation is one that, on every input, produces a solution whose cost is $\leq \alpha OPT$, where OPT is the optimal cost.

As mentioned, this problem is NP-complete, but we can actually say something stronger. There exists some specific constant c such that if one can find a polynomial-time $c \ln(n)$-approximation, then $P = NP$.

Audience Question: Why is this bound so bad? Meaning, why can't we hope to get a closer approximation to the optimal solution?

Answer: The landscape of approximation is very difficult. The computer science theory community has discovered that different NP-complete problems have very different approximation bounds. For example, if the Traveling Salesman Problem is constrained so that all destinations lie in the plane, it is NP-complete but there are algorithms that achieve $1+\epsilon$-approximation, so very close to optimal. On the other hand, there are NP-complete problems where the best known algorithms are $O(n^c)$-approximations, even worse than set cover. So in summary, not all NP-complete problems are created equal when it comes to approximation.

2.2 Step 1: Integer Program

The first step to finding the approximate solution to this problem is to formulate it as an integer program:

Input: $e_1, \ldots, e_n$, $S_1, \ldots, S_m$, $w_1, \ldots, w_m$

Declare $z_j \in \{0, 1\}$: Is S_j in the cover or not?

Objective: Minimize the weights of sets we pick, $\sum_{j=1}^{m} w_j z_j$

Constraints: For every element,at least one set that covers it must be included: $\sum_{j|e_i \in S_j} z_j \geq 1$

Define OPT to be the solution to this IP, which we want to approximate.

2.3 Step 2: Linear Program

Now relax to a linear program, specifically this means everything is the same except for one change, z_j is real-valued such that:

$0 \leq z_j \leq 1$

Intuitively, one can choose to partially include a set in the cover instead of making a hard choice.

2.4 Step 3: Solve LP

Now, we can solve this LP in polynomial time. Let V^* denote the value of the optimal LP solution.

We know the Linear Program has a better or equal value to the IP, since any IP solution can be represented as an LP. So, remembering this is a minimization problem:

$OPT \geq V^*$

2.5 Step 4: Round LP Solution

Now we have a solution to the LP: $z_1^*, \ldots, z_m^*$, and the problem at hand is how to convert them back to integers. Intuitively, the higher the z_i^*, the more we want S_i to be in the cover.

The key idea is to include s_j with probability z_j^*, essentially tossing a biased coin to determine whether to include each subset.

This has a really convenient property, that if we let:

$$z_j = \begin{cases} 1 & : s_j \text{ selected} \\ 0 & : \text{otherwise} \end{cases}$$

Then $\mathbb{E}[z_j] = z_j^*$. So $\mathbb{E}\left[\sum_{j=1}^{m} w_j z_j\right] = \sum_{j=1}^{m} w_j \mathbb{E}[z_j] = \sum_{j=1}^{m} w_j z_j^* = V^* \leq OPT$

So we have shown that we expect to get a solution as good as the optimal solution!

2.6 Step 5: Analysis of the Algorithm

But wait, this selection of subsets might not be a solution, because it might not actually cover all the elements. Let's compute the probability some element wasn't covered:

$P(e_i \text{ not covered}) = \prod_{j|e_i \in S_j} (1 - z_j^*)$

By our favorite inequality, that $\prod_i (1-x_i) \leq e^{-\sum_i x_i}$, we can say:
$\prod_{j|e_i \in S_j}(1-z_j^*) \leq e^{-\sum_{j|e_i \in S_j} z_j^*}$
And since the sum of the z_j^*, must be at least one, we can bound this by $P(e_i \text{ not covered}) \leq \frac{1}{e}$. So in other words, we didn't cover a constant fraction of the elements, which isn't good enough.

To fix this, we can try repeating the process k times. By this we mean that we take the sets chosen in the second round and add them to the first round, etc. until we have a set cover. In that case, $P(e_i \text{ not covered}) \leq \frac{1}{e^k}$. So in particular, if we let $k = c \ln n$, we can get this probability down to $\leq \frac{1}{n^c}$.
However, what we really want is not $P(e_i \text{ not covered})$, but $P(\exists e_i \text{ not covered})$
$P(\exists e_i \text{ not covered}) = P(e_1 \text{ not covered} \vee e_2 \text{ not covered} \vee \cdots \vee e_n \text{ not covered})$
There are complicated dependencies that we don't understand, so we have to break this disjunction up using the union bound:
$P(\exists e_i \text{ not covered}) = P(e_1 \text{ not covered} + e_2 \text{ not covered} + \cdots + e_n \text{ not covered})$
$\leq n\frac{1}{n^c} \leq \frac{1}{n}$for $c = 2$.

Now we just need to find the expected cost of the resulting sets. To do so, we define:

$$Z_j^l = \begin{cases} 1 & : s_j \text{ selected on } l^{th} \text{ repetition} \\ 0 & : \text{otherwise} \end{cases}$$

Then, we know, for some l, $\mathbb{E}\left[Z_j^l\right] = z_j^*$, so if we do $2 \ln n$ iterations:
$\mathbb{E}\left[\text{cost of resulting collection of sets}\right] = \mathbb{E}\left[\sum_{l=1}^{2\ln n}\sum_{j=1}^{m} w_j Z_j^l\right] = 2\ln n \sum_{j=1}^{m} w_j z_j^* = 2 \ln n V^* \leq 2 \ln n \, OPT.$
So, we have found a randomized algorithm which produces a result with cost $\leq c \ln n OPT$ and which is a solution with probability $\geq 1 - \frac{1}{n}$. However, we are not quite done yet. We want to restrict the expected cost calculation to only include set covers.

So, if C denotes the cost and SC indicates that we have a set cover, we want $\mathbb{E}\left[C|SC\right]$:
$\mathbb{E}\left[C\right] = \mathbb{E}\left[C|SC\right] P(SC) + \mathbb{E}\left[C|\neg SC\right] P(\neg SC)$
$\mathbb{E}\left[C|SC\right] P(SC) = \mathbb{E}\left[C\right] - \mathbb{E}\left[C|\neg SC\right] P(\neg SC)$
$\mathbb{E}\left[C|SC\right] P(SC) \leq \mathbb{E}\left[C\right]$
$\mathbb{E}\left[C|SC\right] \leq \frac{\mathbb{E}[C]}{P(SC)} \leq \frac{2 \ln n V^*}{1-\frac{1}{n}}$
Since we assume $n \geq 2$, $1 - \frac{1}{n} \geq \frac{1}{2}$. Thus, $\frac{2 \ln n V^*}{1-1/n} \leq 4 \ln n V^*$.
So far, we have shown the following theorem.

Theorem 1. *The randomized algorithm produces set cover with expected cost at most $c \ln n OPT$ and with probability at least $1 - 1/n$.*

3 Example 2: Max SAT

3.1 Problem Statement

In this problem, we have a set of n boolean variables $x_1, \ldots, x_n$. Next we have a set of clauses $c_1, \ldots, c_m$ which are disjunctions of variables or their negations. We also have a set of weights $w_1, \ldots, w_m$, one for each clause. The goal is to find the setting of variables that maximizes the weight of the satisfied clauses.

3.2 Step 1: Integer Program

Define $y_i \in \{0, 1\}$, as is the variable x_i true or false. Define $z_j \in \{0, 1\}$ as is clause c_j satisfied.
The objective of the IP is to maximize$\sum_j = 1^m w_j z_j$.
Define P_j as the set of variables in c_j that occur without negation, and N_j as those that occur with negation. Then the IP constraints are defined as, for all j from 1 to m:

$$\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j$$

This serves to constrain the z_j's to only be 1 if clause j was satisfied. If c_j is unsatisfied, since it is a disjunction that means $\sum_{i \in P_j} = 0$ and $\sum_{i \in N_j} (1 - y_i) = 0$, so z_j must be zero. If it is satisfied, one or more variables (or negations) must be 1, making the sum at least one.

3.3 Step 2: Relax to LP

The relaxation of the integer to linear program can be achieved simply by setting y_i and z_i to real values, $0 \leq y_i \leq 1$, $0 \leq z_i \leq 1$.

3.4 Step 3: Solve the LP

This LP can be solved to get an optimal cost $V^* \geq OPT$, and optimal settings y_i^*, and z_i^*.

3.5 Step 4: Round to original solution

To randomly round, we can set the variables as follows:

$$x_i = \begin{cases} 1 & : \text{with probability } y_i^* \\ 0 & : \text{otherwise} \end{cases}$$

3.6 Step 5: Analysis of the Algorithm

We will show below that the expected weight of this scheme is at least $1 - \frac{1}{e}$.

Theorem 2. *The algorithm above gives* $1 - 1/e$ *approximation ratio.*

Clearly, the expected weight of the result is $\sum_{j=1}^{m} w_j P(C_j \text{ satisfied})$

We can use the y_i^* to find the probability a clause is not satisfied be separating the variables into positive and negated, as we did for the IP constraints:

$$P(C_j \text{ not satisfied}) = \prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} (y_i^*)$$

This is pretty hard to work with, we would like some way to turn this product into a sum. One way would be to use our favorite inequality, that $1 - x \leq e^{-x}$, but it turns our that would give us too loose of a bound. An alternative method is to use the Arithmetic/Geometric Mean inequality, which states, assuming the a_i are nonnegative:

$$(\prod_{i=1}^{k} a_i)^{\frac{1}{k}} \leq \frac{1}{k} \sum_{i=1}^{k} a_i$$

Below we include a proof of this inequality.

Lemma 3. $(\prod_{i=1}^{k} a_i)^{\frac{1}{k}} \leq \frac{1}{k} \sum_{i=1}^{k} a_i$ *for nonnegative* a_i*'s.*

Proof

Case 1: Some a_i is zero. In this case, since we include a zero in the product $(\prod_{i=1}^{k} a_i)^{\frac{1}{k}} = 0$ and $\frac{1}{k} \sum_{i=1}^{k} a_i \geq 0$, so the statement holds.

Case 2: All a_i are strictly positive.

First we need to show that the function $f(x) = \ln x$ is both monotonically increasing and concave. To show it is **monotonically increasing** we take the first derivative, $f'(x) = \frac{1}{x}$. Now, x must be positive, since the natural logarithm is only defined over a positive domain. And if x is positive then so is $\frac{1}{x}$. To show it is **concave** we take the second derivative $f''(x) = -\frac{1}{x^2}$. If x is postive, $-\frac{1}{x^2}$ will be ngative, so $f(x)$ is concave.

Now, Jensen's inequality tells us that for any convex function f,

$$f(\sum_i \lambda_i x_i) \leq \sum_i \lambda_i f(x_i)$$

Where the λ_i are nonnegative and sum to 1. So since we proved ln is concave, $-\ln$ is convex and thus:

$$\begin{aligned}
-\ln(\sum_i \lambda_i a_i) &\leq \sum_i -\lambda_i \ln(a_i) \\
\ln(\sum_i \lambda_i a_i) &\geq \sum_i \lambda_i \ln(a_i) \\
\ln(\sum_i \lambda_i a_i) &\geq \sum_i \ln(a_i^{\lambda_i}) \\
\ln(\sum_i \lambda_i a_i) &\geq \ln(\prod_i a_i^{\lambda_i})
\end{aligned}$$

Now, $f(x) = \ln x$ is a strictly increasing function, which in particular means that if $f(x_1) \geq f(x_2)$, then $x_1 \geq x_2$. So we can remove the ln's from the inequality:

$$\sum_i \lambda_i a_i \geq \prod_i a_i^{\lambda_i}$$

In particular, if we let $\lambda_i = \frac{1}{k}$ (satisfying the requirements of being nonnegative and summing to 1):

$$\begin{aligned}
\sum_{i=1}^{k} \frac{1}{k} a_i &\geq \prod_{i=1}^{k} a_i^{\frac{1}{k}} \\
\frac{1}{k} \sum_{i=1}^{k} a_i &\geq \prod_{i=1}^{k} a_i^{\frac{1}{k}}
\end{aligned}$$

Note that $\frac{1}{k}$ does not depend on i, and in general $\prod_i x_i^c = (\prod_i x_i)^c$ since there are the same number of x_i terms in the resulting product. So we can move the exponential outside the product as follows:

$$\frac{1}{k} \sum_{i=1}^{k} a_i \geq (\prod_{i=1}^{k} a_i)^{\frac{1}{k}}$$

■

Because of thus inequality, we can show, where l_j is the number of literals in clause j:

$$[\prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} (y_i^*)]^{\frac{1}{l_j}} \leq \frac{1}{l_j} (\sum_{i \in P_j} (1 - y_i^*) + \sum_{i \in N_j} y_i^*)$$

Which tells us:

$$\prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} (y_i^*) \leq [\frac{1}{l_j} (\sum_{i \in P_j} (1 - y_i^*) + \sum_{i \in N_j} y_i^*)]^{l_j}$$

We can rewrite this as follows:

$$= [1 - \frac{1}{l_j}(\sum_{i \in P_j}(y_i^*) + \sum_{i \in N_j} 1 - y_i^*)]^{l_j}$$

Remember that in our LP, we had said

$$\sum_{i \in P_j} y_i^* + \sum_{i \in N_j}(1 - y_i^*) \geq z_j^*$$

, so:

$$\leq (1 - \frac{z_j^*}{l_j})^{l_j}$$

Therefore,

$$P(C_j \text{ satisfied}) \geq 1 - (1 - \frac{z_j^*}{l_j})^{l_j}$$

However, having the z_j^* hidden inside this expression is cumbersome. We need some way to bring it into a more accessible location. To do this, the first step is to observe that the function $f(x) = 1 - (1 - \frac{x}{l})^l$ is concave when x ranges from 0 to 1. To formally prove this, we need to look at the second derivative and show that it is always negative in that range:

$$f'(x) = -l(1 - \frac{x}{l})^{l-1}(-\frac{1}{l}) = (1 - \frac{x}{l})^{l-1}$$

$$f''(c) = (l-1)(1 - \frac{x}{l})^{l-2}(-\frac{l}{l}) = (-\frac{l-1}{l})(1 - \frac{x}{l})^{l-2}$$

Here it should be clear that if x is between 0 and 1, $(1 - \frac{x}{l})^{l-2}$ will always be nonnegative, and thus $f''(c) \geq 0$.

Hence, the function is concave and we can lower bound it with a line in the domain $[0, 1]$. To determine the endpoints of the line, observe that $f(0) = 0$ and $f(1) = 1 - (1 - \frac{1}{l})^l$. Hence, the equation for the line passing through these two points is simply $g(x) = [1 - (1 - \frac{1}{l})^l]x$, and we know $f(x) \geq g(x)$.

Returning to the problem at hand, we can now say:

$$P(C_j \text{ satisfied}) \geq 1 - (1 - \frac{z_j^*}{l_j})^{l_j} \geq [1 - (1 - \frac{1}{l_j})^{l_j}]z_j^*$$

So now,

$$\text{Expected weight of result} = \sum_{j=1}^{m} w_j P(C_j \text{ satisfied}) \geq \sum_{j=1}^{m} w_j z_j^* [1 - (1 - \frac{1}{l_j})^{l_j}]$$

Now, we recognize that $\sum_{j=1}^{m} w_j z_j^*$ is V^*. Since we are trying to lower bound this function,

$$\geq V * \min_j [1 - (1 - \frac{1}{l_j})^{l_j}$$

We know that, using our useful approximation:

$$(1 - x)^y \leq e^{-xy}$$

$$(1 - \frac{1}{k})^k \leq e^{-k\frac{1}{k}} = e^{-1}$$

Therefore:

$$\text{Expected weight of result} \geq V * (1 - \frac{1}{e})$$

4 Integrality Gap

One question is, can we do better than this using this method? To help answer this question, we can introduce the notion of an "integrality gap", that is, what is the smallest we can expect the gap between the optimal solution to the integer and linear program to be? In other words,

$$\text{Integrality gap} = \lim_{\text{numInstances} \to \infty} \frac{\text{IP optimum of I}}{\text{LP optimum of I}}$$

We can upper bound this with the following example:

$c_1 = x_1 \vee x_2,\ c_2 = x_1 \vee \neg x_2,\ c_3 = \neg x_1 \vee x_2,\ c_4 = \neg x_1 \vee \neg x_2$. All have weight 1.

The optimal solution to the integer program here is 3, because for any setting of x_1, x_2, the negation of both will not hold. For example, if $x_1 = 1$ and $x_2 = 0$, then c_3 will not hold but the other three will.

However, with a linear program, we can let $x_1 = 0.5, x_2 = 0.5$ and all four will hold.

This shows that for this problem, the integrality gap is $\leq \frac{3}{4}$. Meaning using this technique, we can't hope to show our discovered solution is better than $\frac{3}{4}$ the LP optimum, even though it may be the optimal value in the IP.

5 Nonlinear Rounding

However, there is still a gap between the $(1 - \frac{1}{e})$ result (less than $\frac{2}{3}$) and the upper bound of $\frac{3}{4}$. Here we show how to do better using a non-linear rounding technique.

Basically, instead of setting x_i with probability y_i^*, we set x_i with probability $f(y_i^*)$.

We want $P(C_j \text{ not satisfied}) \geq \frac{3}{4} z_j^* = (1 - \frac{1}{4}) z_j^*$.

Now, observe that since $(1 - 4^{-x})$ is concave in the domain $[0, 1]$ we can use it to upper bound the line formed by $(1 - \frac{1}{4})x$.

So it suffices to have $P(C_j \text{ satisfied}) \geq 1 - 4^{-z_j^*} \geq (1 - \frac{1}{4}) z_j^*$.

Recall that

$$P(C_j \text{ not satisfied}) = \prod_{i \in P_j} (1 - f(y_i^*)) \prod_{i \in N_j} f(y_i^*)$$

And we want that to be:

$$\leq 4^{-[\sum_{i \in P_j} y_i^* + \sum_{i \in N_j} (1 - y_i^*)]} = 4^{-z_j^*}$$

To do that, it suffices to have $1 - f(x) \leq 4^{-x}$ and $f(x) \leq 4^{-(1-x)}$

We can rewrite this as $1 - 4^{-x} \leq f(x) \leq 4^{x-1}$

Looking at figure 1, on the interval $[0, 1]$, $1 - 4^{-x} \leq 4^{x-1}$.

Of course, the figure is not a proof. To formally prove this, we need to show:

$$1 - 4^{-x} \leq 4^{x-1}$$

$$1 \leq 4^{-x} + 4^{x-1}$$

So it is sufficient to show that the minimum value of $4^{-x} + 4^{x-1}$ is 1. Let $f(x) = 4^{-x} + 4^{x-1}$, then $f'(x) = -\ln(4)4^{-x} + \ln(4)4^{x-1}$. Now we set $f'(x) = 0$ to find the critical point, and we have

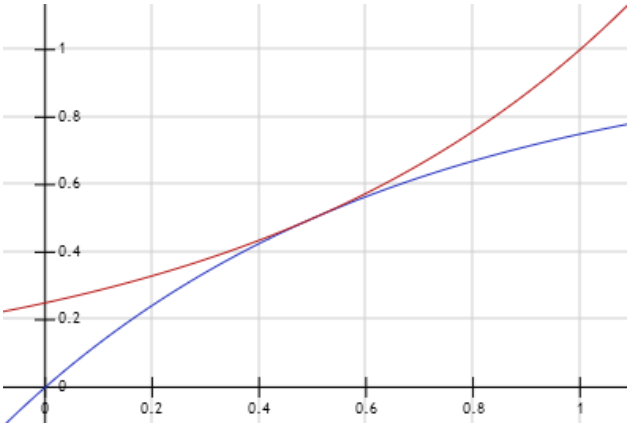

Figure 1: A graph comparing $1 - 4^{-x}$ (in blue) and 4^{x-1} (in red). As one can see $1 - 4^{-x} \leq 4^{x-1}$ in the domain $[0, 1]$.

$$0 = -\ln(4)4^{-x} + \ln(4)4^{x-1}$$

$$\ln(4)4^{-x} = \ln(4)4^{x-1}$$

$$-x = x - 1$$

$$x = \frac{1}{2}$$

So $f'(x)$ has a single critical point at $c = \frac{1}{2}$. We can determine whether this is a absolute maximum or minimum over the interval $[0, 1]$ by evaluating f at the edges of the interval and the critical point: $f(0) = \frac{5}{4}$, $f(c) = 1$, $f(1) = \frac{5}{4}$. So 1 is the absolute minimum of f in the interval $[0, 1]$, and thus $1 - 4^{-x} \leq 4^{x-1}$.

Hence, we can let $f(x) = 4^{x-1}$ and satisfy the inequality. With this choice, $\mathbb{E}[\text{weight of solution}] = \sum_{j=1}^{m} w_j P(C_j \text{ satisfied}) \geq \frac{3}{4} \sum_{j=1}^{m} w_j z_j^* \geq \frac{3}{4} OPT$.