CS 388R: Randomized Algorithms, Fall 2023 November 29, 2023

Lecture 26: Randomized Rounding

Prof. Eric Price *Scribe: Sameer Gupta, Mark Wen*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Set Cover

Suppose we have some universe U of n items. We have some sets $S_1, S_2, \ldots, S_m \subseteq U$, weighted as $w_1, w_2, \ldots, w_m > 0$. A set of indices $C \subseteq [m]$ is a ***cover*** if $\bigcup\limits_{i \in C} S_i = U$. The cost of a cover is $\sum\limits_{i \in C} w_i$. Our goal is to find a cover with minimum cost.

One example is edge cover, which asks us to find a set of edges that covers every vertex. Similarly, a vertex cover is a set of vertices that covers every edge. Both of these problems can be represented as set covers. Vertex cover is NP-hard, so set cover must also be NP-hard. Thus, we want to find some approximation of the answer. Specifically, the cost should be at most some multiplicative factor α above the optimal. Formally, find some cover C such that $\text{cost}(C) \leq \alpha \cdot OPT$ where OPT is the min cost cover.

For edge cover, a trivial lower bound is $\frac{V}{2}$. We can greedily pick some edge next to each vertex, and this gives us a cover of at most $V - 1$. Thus, we get a 2-approximation. This strategy works because the sizes of the sets in the cover are small.

2 Randomized Rounding

Randomized rounding helps us solve more general problems. First, we write the proble mas an integer linear program. We then relax the constraints to fractional linear programs, and then we round to an integer solution. By rounding, we lose some factor from the optimum.

Since we fractional LP contains a superset of the solution space of an integer linear program, we know $\alpha \cdot OPT_{LP} \leq \alpha \cdot OPT$, so if we can find a C such that $\text{Cost}(C) \leq \alpha \cdot OPT_{LP}$, then we are good

To represent set cover as a linear program, we say $x_i \in \{0, 1\}$, where $x_i = 1$ if $i \in C$. We are trying to minimize the sum of weights $\sum w_i x_i$, and our constraints are $\sum\limits_{i: j \in S_i} x_i \geq 1 \, \forall j \in U$.

To convert this to a general linear program, we allow $x_i \in [0, 1]$. We can then use some polynomial time algorithm to find the solution quickly. The optimum found by linear programs will be less than the integer linear program optimum, which is the true optiumum.

As an example for where rounding doesn't work, suppose we have three vertices in a triangle. The minimum edge cover is 2, but the linear program gives an output of $\frac{3}{2}$ by assigning $\frac{1}{2}$ to each edge.

To round our solution without running into edge cases, the idea is to randomly round the fraction. One approach is to round to $\hat{x}_i = 1$ with probability x_i and to $\hat{x}_i = 0$ with probability $1 - x_i$, independently for all i. We don't want to just round to the closest integer. Suppose we have a single edge covered by multiple sets. The linear program output might assign a small weight to each set, and so it would round each of the values down. By randomly rounding, there is a decent chance that at least once of the indices will be included.

To bound the probability that this gives a valid solution, the probability of some element j being uncovered is

$$\begin{aligned}\prod_{i:j\in S_i}(1-x_i) &\leq \prod_{i:j\in S_i} e^{-x_i}\\ &= \exp\left(\sum_{i:j\in S_i} -x_i\right)\\ &\leq 1/e.\end{aligned}$$

The expected cost is close to optimal to the LP, but the chance of some vertex being uncovered is actually pretty high.

Now, suppose we repeat this T times, and let $\hat{x}_i = 1$ if *any* of the roundings included S_i. The expected cost is at most $T \cdot \mathrm{OPT}_{\mathrm{LP}}$, and the probability j is uncovered is e^{-T}. If we set $T = \log(4n)$, then the probability any element is uncovered is at most $\frac{1}{4}$. By Markov's inequality, $\mathbb{P}[\mathrm{cost}(\hat{x}) \geq 4T\,\mathrm{OPT}_{\mathrm{LP}}] \leq \frac{1}{4}$. Thus, at least 50% of the time, we get a cover of cost at most $4T\,\mathrm{OPT}_{\mathrm{LP}}$. It is actually NP-hard to get better than a $\log n$ approximation for set cover.

3 Max-SAT

In Max-SAT, we have a series of m clauses $C_j = (C_j^+, C_j^-)$, with length ℓ_j. We have n variables in total. Our question is to find the maximum number of satisfiable clauses at the same time. This is harder than 3-SAT because we can just directly represent 3-SAT in this problem and check if the maximum equals m, so this is NP-hard.

To try to solve the 3-SAT case, we can just randomly set x_i to uniform random values. The chance that a clause is unsatisfiable is $\frac{1}{8}$, so the expected number of satisfied clauses is $\frac{7}{8}$. Thus, we expect a $\frac{7}{8}$-approximation.

The issue is that we have no bounds on how this concentrates around the mean. One way to fix this is to do derandomization. We first try $x_1 = 0$ and $x_1 = 0$, and see the expected number of satisfied clauses. We can then compute the expectation given a fixed value of x_1, and we choose the branch that gives us at least $\frac{7}{8}$ in expectation. Doing this repeatedly gives us at least an $\frac{7}{8}$ approximation.

In general, as we increase the clause size, we get closer to the optimum. Our expectation is actually $\sum_j (1-2^{-\ell_j})$. As long as $\ell_j \geq 3$, this works well. However, the expectation is low for smaller values, and we would expect that this shouldn't be an issue since Max-2-SAT is solvable. To address this issue, we construct a linear program for the short clauses and do randomized rounding.

Let y_j for $j \in [m]$ represent if the jth clause is satisfied, and let x_i for $i \in [n]$ represent if the ith variable is 1. We are trying to maximize $\sum y_j$, and our constraints are

$$y_j \leq \left(\sum_{i \in C_j^+} x_i\right) + \left(\sum_{i \in C_j^-} 1 - x_i\right).$$

We also have $x_i, y_j \in \{0,1\}$ in the integer variant, but we relax this to $x_i, y_j \in [0,1]$ so that we can solve it. Assume the original expression was fully satisfiable, which means the optimal is $y_j = 1$ for everything. Given a fractional solution to SAT, we try setting $\hat{x}_i = 1$ with probability x_i. We now measure the probability that some clause is not satisfied, which is

$$\begin{aligned}\left(\prod_{i \in C_j^+} (1 - x_i)\right)\left(\prod_{i \in C_j^-} (1 - (1 - x_i))\right) &\leq \prod_{i \in C_j^+} e^{-x_i} \prod_{i \in C_j^-} e^{-(1-x_i)} \\ &= \exp\left(-\sum_{i \in C_j^+} x_i - \sum_{i \in C_j^-} (1 - x_i)\right) \\ &\leq e^{-1} \\ &= 1/e.\end{aligned}$$

Thus, the expected number of satisfied clauses is $\left(1 - \frac{1}{e}\right)m \approx 0.63m$. This is not a very good bound. To improve this, we can look at the length of the list as well. With this, we have

$$\begin{aligned}\left(\prod_{i \in C_j^+} (1 - x_i)\right)\left(\prod_{i \in C_j^-} (1 - (1 - x_i))\right) &\leq \left(\frac{\left(\sum_{i \in C_j^+} (1 - x_i)\right) + \left(\sum_{i \in C_j^-} (1 - (1 - x_i))\right)}{\ell_j}\right)^{\ell_j} \\ &= \left(1 - \frac{\left(\sum_{i \in C_j^+} x_i + \sum_{i \in C_j^-(1-x_i)}\right)}{\ell_j}\right)^{\ell_j} \\ &\leq \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}.\end{aligned}$$

Now, we can compute the probability of failure depending on the length. We get

x_j	independent random	rounded LP	random choice
ℓ	$1 - 2^{-\ell}$	$1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}$	average
1	.5	1	.75
2	.75	.75	.75
3	.875	.7	.82
4	.94	.683	.83

By combining both of the strategies, we have a higher chance of success than if we just used one of them. The first probability approaches 1 and the second probability approaches $1/e$, so the average is about .81.

Suppose we don't know that the expression is fully satisfiable. We want to compare the proability of failure, which is now $\left(1 - \frac{y_j}{\ell_j}\right)^{\ell_j}$, with y_j itself. Specifically, we are comparing $\sum_j \left(1 - \frac{y_j}{\ell_j}\right)^{\ell_j}$ with $\sum y_j$. If $\ell = 2$, then we have

$$\begin{aligned}\sum 1 - \left(1 - \frac{y_j}{2}\right)^2 &= \sum y_j - \frac{y_j^2}{4} \\ &= \sum y_j \cdot \left(1 - \frac{y_j}{4}\right) \\ &\geq \frac{3}{4} \sum y_j.\end{aligned}$$

Thus, the ratio works out for this case.