Algorithms II October 6, 2025

Lecture 8: Approximation Algorithms using LPs

Notes by Ola Svensson[1]

In this lecture we do the following:

- We give a randomized approximation algorithm for the Set Cover problem
- We show that the integrality gap of the set cover LP is $\Omega(\log n)$

These notes are based on [1] and [2].

1 Set Cover via Randomized Rounding

Let us now apply the framework to the *Set Cover* problem. It can be seen as a generalization of the vertex cover problem and its definition is as follows:

Definition 1 (Set Cover Problem) *Given a universe $\mathcal{U} = \{e_1, e_2, \dots e_n\}$, and a family of subsets $\mathcal{T} = \{S_1, S_2, \dots S_m\}$ and a cost function $c : \mathcal{T} \to \mathbb{R}_+$, find a collection C of subsets of minimum cost that cover all elements.*

As for vertex cover, we start by giving an exact Integer LP formulation. For each $i \in \{1, \dots m\}$, define x_i, which is 1 if $S_i \in C$, and 0 otherwise. The objective function is

$$\min \sum_{i=1}^{m} x_i \cdot c(S_i)$$

and for each element $e \in \mathcal{U}$, we add the constraint $\sum_{S_i \,:\, e \in S_i} x_i \geq 1$. This ensures that each element is covered by at least one set in C. And for each x_i, we require that $x_i \in \{0, 1\}$ in the ILP. The LP relaxation is then obtained by replacing the boolean constraints $x_i \in \{0, 1\}$ by $x_i \in [0, 1]$.

Now suppose that each element belongs to at most f sets. Then, as in your exercise on vertex cover on k-uniform hypergraphs, we can do the following rounding: $C = \{S_i : x_i^* \geq \frac{1}{f}\}$. In each constraint, there's at least one x_i^* which is at least $\frac{1}{f}$, so each constraint is satisfied. Using the same reasoning as in the analysis of the vertex cover rounding, we can show that this approximation is within a factor of f.

1.1 A better approximation for Set Cover

If we introduce randomness and allow our algorithm to output non-feasible solutions with some small probability, we can get much better results (in expectation).

We use the same LP as in the previous section, and will run the following algorithm:

1. Solve the LP to get an optimal solution x^*.
2. Choose some positive integer constant d (we will see later how d affects the guarantees we get). Start with an empty result set C, and repeat step 3 $d \cdot \ln(n)$ times.
3. For $i = 1, \dots m$, add set S_i to the solution C with probability x_i^*, choosing independently for each set.

Now let us analyze what guarantees we can get:

[1] **Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

Claim 2 *The expected cost of all sets added in one execution of Step 3 is*

$$\sum_{i=1}^{m} x_i^* c(S_i) = LP_{OPT}$$

Proof

$$\mathbb{E}[\text{rounded cost}] = \sum_{i=1}^{m} c(S_i) \Pr[S_i \text{ is added}] = \sum_{i=1}^{m} c(S_i) x_i^* = LP_{OPT}$$

■

From this, we can immediately derive

Corollary 3 *The expected cost of C after $d \cdot \ln(n)$ executions of Step 3 is at most*

$$d \cdot \ln(n) \cdot \sum_{i=1}^{m} c(S_i) x^* \leq d \cdot \ln(n) \cdot LP_{OPT} \leq d \cdot \ln(n) \cdot OPT$$

Note that we have $LP_{OPT} \leq OPT$ because LP is a relaxation of the original problem, so its optimum can only be better.

That sounds good, but we should also worry about feasibility:

Claim 4 *The probability that a constraint remains unsatisfied after a single execution of Step 3 is at most $\frac{1}{e}$.*

Proof Suppose our constraint contains k variables, and let us write it as $x_1 + x_2 + \cdots + x_k \geq 1$. Then,

$$\begin{aligned}
\Pr[\text{constraint unsat.}] &= \Pr[S_1 \text{ not taken}] \ldots \Pr[S_k \text{ not taken}] \\
&= (1 - x_1^*) \ldots (1 - x_k^*) \\
&\leq e^{-x_1^*} \cdot \ldots \cdot e^{-x_k^*} \qquad (1) \\
&= e^{-\sum_{i=1}^{k} x_i^*} \\
&\leq e^{-1} \qquad (2)
\end{aligned}$$

where (1) follows from the inequality $1 - x \leq e^{-x}$ and (2) from the fact that $\sum_i x_i^* \geq 1$. ■

Claim 5 *The output C is a feasible solution with probability at least $1 - \frac{1}{n^{d-1}}$.*

Proof Using claim 4, we find that the probability that a given constraint is unsatisfied after $d \cdot \ln(n)$ executions of step 3 is at most

$$\left(\tfrac{1}{e}\right)^{d \cdot \ln(n)} = \frac{1}{n^d}$$

and by union-bound, the probability that there exists any unsatisfied constraint is at most

$$n \cdot \frac{1}{n^d} = \frac{1}{n^{d-1}}$$

■

Now we have an expected value for the cost, and also a bound on the probability that an infeasible solution is output, but we still might have a bad correlation between the two: It could be that all feasible outputs have a very high cost, and all infeasible outputs have a very low cost.

The following claim deals with that worry.

Claim 6 *The algorithm outputs a feasible solution of cost at most* $4d\ln(n)OPT$ *with probability greater than* $\frac{1}{2}$.

Proof Let μ be the expected cost, which is $d\ln(n) \cdot OPT$ by corollary 3. We can upper-bound the bad event that the actual cost is very high: By Markov's inequality, we have $\Pr[cost > 4\mu] \leq \frac{1}{4}$. The other bad event that we have to upper bound is that the output is infeasible, and by claim 5, we know that this happens with probability at most $\frac{1}{n^{(d-1)}} \leq \frac{1}{n}$. Now in the worst case, these two bad events are completely disjoint, so the probability that no bad event happens is at least $1 - \frac{1}{4} - \frac{1}{n}$, and if we suppose that n is greater than 4, this probability is indeed greater than $\frac{1}{2}$. ■

We have thus designed a randomized $O(\log n)$-approximation algorithm for the set cover problem.

We remark that the used framework has the following general advantage (compared to worst-case guarantees): we can often get better per-instance guarantee than the general approximation factor: Suppose we have an instance where $LP_{OPT} = 100$, and our algorithm found a solution of cost 110. Since we know that $LP_{OPT} \leq OPT$, we can say that our solution on this instance is at most 10% away from the optimal solution for this instance.

2 Integrality gap of the set cover LP

Consider the following instance of the Set Cover problem. For an even integer $d \geq 1$ let

$$U = \left\{ x \in \{0,1\}^d : \sum_{i=1}^{d} x_i = d/2 \right\},$$

i.e., the universe consists of all binary vectors of length d that have $d/2$ nonzeros. Let the collection $\mathcal{F}$ contain $m = d$ sets $S_1, \ldots, S_m$, defined by

$$S_i = \{x \in U : x_i = 1\}$$

for every $i = 1, \ldots, m$. All costs are 1.

We first give a feasible solution to the LP relaxation of Set Cover on the instance above with value bounded by 2. The LP relaxation of the set cover problem is the following:

$$\begin{aligned} &\min_z \sum_{i=1}^{d} z_i, \text{ st.:} \\ &\forall i \in [d] : z_i \in [0,1] \\ &\forall x \in U : \sum_{i:x \in S_i} z_i \geq 1 \end{aligned}$$

A solution to this with value 2 is to set every variable z_i to $2/d$. That way $\sum_i z_i = 2$ and all constraints are satisfied:

$$\sum_{i:x \in S_i} z_i = \sum_{i:x_i=1} z_i = \sum_{i:x_i=1} 2/d = \sum_{i=1}^{n} 2/d \cdot x_i = 1$$

Suppose we have any collection of $d/2$ sets $\mathcal{F}' \subseteq \mathcal{F}$. We can characterise $\mathcal{F}'$ as $\{S_i : i \in \mathcal{I}\}$ for some $\mathcal{I} \subseteq \{1, \ldots, d\}$ with $|\mathcal{I}| = d/2$. Let us then define the vector x^* such that $x^*_i = 0$ for $i \in \mathcal{I}$ and $x^*_i = 1$ for $i \notin \mathcal{I}$. Then $x^* \in U$ but $x^* \notin \cup\mathcal{F}$, thus proving that $\mathcal{F}$ does not cover U. In this set cover problem the optimal integral solution is at least $d/2 + 1$, but the optimal fractional solution is at most 2. This is an $\Omega(d)$ integrality gap. Since the size of the universe, $|U| = \binom{d}{d/2} \leq 2^d$, this translates to an $\Omega(\log |U|)$ integrality gap.

3 The Multiplicative Weights Algorithm

This section is basically taken verbatim from the excellent lecture notes[2] by Anupam Gupta available here (together with a lot of other interesting information): *https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/*

Multiplicative weights is a retronym for the simple iterative rule that underlies modular, iterative algorithms for solving LPs and SDPs (semidefinite programs)[3]. The Multiplicative Weights Algorithm is known by diferent names in the various fields where it was (re)discovered. Check out the survey by Arora, Hazan and Kale [3]; our discussion will be based on their treatment. Due to its broad appeal, we will consider multiplicative weights in more generality than is needed for solving LPs and SDPs. In this lecture, we'll introduce some strategies for playing a prediction game.

3.1 Warmup: Prediction with Expert Advice

The following sequential game is played between an omniscient Adversary and an Aggregator who is advised by N experts. Special cases of this game include predicting if it will rain tomorrow, or if the stock market will go up or down.

Strategy 1 Prediction with Expert Advice

1: **for** $t = 1 \dots T$ **do**
2: Each expert $i \in [N]$ advises either YES or NO
3: Aggregator predicts either YES or NO
4: Adversary, with knowledge of the expert advice and Aggregator's prediction, decides the YES/NO outcome.
5: Aggregator observes the outcome and suffers if his prediction was incorrect.
6: **end for**

Naturally, Aggregator wants to make as few mistakes as possible. Since the experts may be unhelpful and the outcomes may be capricious, Aggregator can hope only for a relative performance guarantee. In particular, Aggregator hopes to do as well as the best single expert in hindsight[4]. In order to do so, Aggregator must track which experts are helpful. We will consider a few tracking strategies. Almost every other aspect of the game - that advise is aggregated into a single value, that this value is binary, and even that the game is sequential - is not relevant; we will generalize or eliminate these aspects.

If there is a perfect expert, then an obvious strategy is to dismiss experts who are not perfect. With the remaining experts, take a majority vote. Whenever the Aggregator makes a mistake, at least half of the remaining experts are dismissed, so Aggregator makes at most $\log N$ mistakes[5]. We can use the same strategy even when there isn't a perfect expert, if we restart after every expert has been eliminated. If the best expert has made M mistakes by time T, then Aggregator has restarted at most $M+1$ times, so it has made at most $(M+1) \cdot \log N$ mistakes. This bound is rather poor since it depends multiplicatively on M.

3.2 Fewer Mistakes with Weighted Majority

We may obtain an additive mistake bound by softening our strategy: instead of dismissing experts who erred, discount their advice. This leads to the *Weighted Majority Algorithm* of Littlestone and Warmuth [4]. Assign each expert i a weight $w_i^{(1)}$ initialized to 1. Then for every t, predict YES/NO based on the weighted majority vote and halve the mistaken experts' weights after observing the outcome. The game strategy then becomes:

[2]This section corresponds to Lecture 16 in the course, which was scribed by Shiva Kaul
[3]See Lecture 17 in the course linked earlier
[4]The excess number of mistakes is called (external) regret
[5]We will use log to denote the binary logarithm

Strategy 2 Prediction with Weighted Majority

1: Initialize $\mathbf{w}^{(1)} = (w_1^{(1)}, \ldots, w_N^{(1)})$ to be a vector of 1's
2: **for** $t = 1 \ldots T$ **do**
3: Each expert $i \in [N]$ advises either YES or NO
4: Aggregator predicts either YES or NO based on a weighted majority vote using $\mathbf{w}^{(t)}$
5: Adversary, with knowledge of the expert advice and Aggregator's prediction, decides the YES/NO outcome.
6: Aggregator observes the outcome and for every mistaken expert i, set $w_i^{(t+1)} \leftarrow w_i^{(t)}/2$
7: **end for**

Claim 7 *For any sequence of outcomes, duration T, let M_{WM} be the number of mistakes that the Weighted Majority strategy makes, and M_i be the number of mistakes that expert i makes. Then*

$$M_{WM} \leq 2.41 \cdot (M_i + \log N)$$

Proof Let

$$\Phi^{(t)} = \sum_{i \in [N]} w_i^{(t)}$$

be a 'potential' function. Observe that:

- By definition, we have $\Phi^{(1)} = N$
- Also by definition, $\left(\frac{1}{2}\right)^{M_i} \leq \Phi^{(T+1)}$
- At any time τ when WM errs, at least half of the weight gets halved:

$$\Phi^{(\tau+1)} \leq \frac{3}{4}\Phi^{(\tau)}$$

This implies

$$\Phi^{(T+1)} \leq \left(\frac{3}{4}\right)^{M_{WM}} \Phi^{(1)}$$

Combining these facts yields

$$\left(\frac{1}{2}\right)^{M_i} \leq \left(\frac{3}{4}\right)^{M_{WM}} N$$

Taking the base-2 logarithm of both sides,

$$-M_i \leq \log N + \log\left(\frac{3}{4}\right) M_{WM}$$

so

$$M_{WM} \leq \underbrace{\frac{1}{\log(4/3)}}_{\approx 2.41}(M_i + \log N)$$

■

The leading constant is a consequence of our arbitrary choice to halve the weights. We can optimize ϵ in the update rule

$$w_i^{(t+1)} \leftarrow w_i^{(t)}/(1+\epsilon)$$

then using the WM strategy we may achieve

$$M_{WM} \leq 2(1+\epsilon)(M_i + O(\log N/\epsilon))$$

References

[1] David Leydier and Samuel Grütter: *Scribes of Lecture 8 in Topics in TCS 2014. http://theory.epfl.ch/courses/topicstcs/Lecture8.pdf*

[2] Romain Edelmann & Florian Tramèr: *Scribes of Lecture 10 in Topics in TCS 2014. http://theory.epfl.ch/courses/topicstcs/Lecture10.pdf*

[3] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16

[4] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In FOCS, pages 256–261, 1989. 16.1.1