Randomization for Approximate Optimization[1]

- **Approximation Algorithms.** Randomization plays an extremely important role in the development of *approximation* algorithms for NP-hard optimization problems. Since we cannot solve NP-hard problems exactly, we often try to find fast algorithms to find solutions which are ***guaranteed*** to be within some specified factor of the best solution. Formally, if our problem Π is a minimization problem, then an α-approximation algorithm (for $\alpha \geq 1$) takes input $I \in \Pi$ and returns a solution whose cost is $\mathsf{cost}(S) \leq \alpha \cdot \mathsf{opt}(I)$. Similarly, if our problem Π is a maximization probem, then an α-approximation algorithm (for $\alpha \geq 1$) takes input $I \in \Pi$ and returns a solution whose "profit" is $\mathsf{profit}(S) \geq \frac{1}{\alpha} \cdot \mathsf{opt}(I)$. When our algorithms are randomized, we let the left hand sides be replaced by $\mathbf{Exp}[\mathsf{cost}(S)]$ and $\mathbf{Exp}[\mathsf{profit}(S)]$, respectively. In this lecture, we look at two basic examples of use of randomization in the design of approximation algorithms.

- **The Set Cover Problem.** The set cover problem is a canonical problem arising is many situations. There is an universe U of n elements. The input consists of m subsets of U namely $S_1, \ldots, S_m$. Each set S_i has a cost $c(S_i)$ or simply c_i. The objective is to pick the ***minimum*** cost family of these sets whose union contains all the elements. More precisely, the objective is to select $A \subseteq [m] = \{1, 2, \ldots, m\}$ such that

$$\text{(a)} \bigcup_{j \in A} S_j = U, \quad \text{and,} \quad \text{(b)} \sum_{j \in A} c(S_j) \text{ is minimized}$$

  This problem is NP-hard, and generalizes the vertex-cover problem which you may have seen in your undergraduate algorithms course. Indeed, you may have even seen this problem itself in your undergraduate algorithms course, and in fact, seen that a "greedy algorithm" gives a good approximation algorithm. Today we use randomization to see another algorithm. But before we do so, we introduce the notion of linear programs.

- **The Linear Program.** One can express the algorithmic question mathematically as follows. Note that our problem is to decide for every subset S_j whether to pick it in A or not. To that end, let us denote this via a vector $\mathbf{x} \in \{0,1\}^m$ where the semantic is $\mathbf{x}_j = 1 \Leftrightarrow j \in A$. Then note, we want to minimize the ***linear function*** $f(\mathbf{x}) = \sum_{j \in [m]} c_j \mathbf{x}_j$, where we have used the short-hand $c_j = c(S_j)$.

  Of course we can't choose any $\mathbf{x} \in \{0,1\}^m$; we have to make sure that the coordinates that are set to 1 correspond to a covering solution. How would we do that? Turns out that this also a ***linear constraint*** on the $\mathbf{x}_j$'s. Fix an element $e \in U$. What do we want? We want that *at least* one set $S_j, j \in A$ should contain this element. In other words, if we look at all the sets which contains this element e, then we must pick at least one of these sets. That is, $\mathbf{x}_j = 1$ for at least one of these sets. Or, the sum of $\mathbf{x}_j$ among the j's such that $e \in S_j$ is at least 1. That is,

$$\text{For all } e \in U, \quad \sum_{j : e \in S_j} \mathbf{x}_j \geq 1$$

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 22nd April, 2021*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Therefore, if we look at the following "program" or an optimization problem

$$\mathrm{P} := \min \sum_{j=1}^{m} c_j \mathbf{x}_j \quad : \quad \text{for all } e \in U, \quad \sum_{j: e \in S_j} \mathbf{x}_j \geq 1, \quad \mathbf{x} \in \{0,1\}^m$$

then, P is a lower bound on the optimum solution of the set-cover instance. For any set-cover solution, we can get an $\mathbf{x} \in \{0,1\}^m$ which satisfies the constraints and whose $f(\mathbf{x})$ equals the cost of the solution. In fact, the value of P is **equal** to the optimum solution, as given any solution $\mathbf{x} \in \{0,1\}^m$, we can obtain a set-cover solution of the same cost. Do you see how?

What progress has been made? We have found a reformulation of the set-cover problem as an equivalent algebraic optimization problem. But since set-cover is NP-hard, solving this algrebraic problem is also NP-hard. The next fact is one of the most important algorithmic achievments. It says that instead of $\mathbf{x} \in \{0,1\}^m$, we has $\mathbf{x} \in [0,1]^m$, then this algebraic problem is ***efficiently solvable***! This formulation is a linear program.

$$\mathrm{LP} := \min \sum_{j=1}^{m} c_j \mathbf{x}_j \quad : \quad \text{for all } e \in U, \quad \sum_{j: e \in S_j} \mathbf{x}_j \geq 1, \quad \mathbf{x} \in [0,1]^m$$

Theorem 1 (Linear Programming)**.** The value LP can be obtained in polynomial time.

Also observe that LP $\leq$ P. This is simply because LP is minimizing over a larger domain. And thus, LP is a ***lower bound*** on the optimum set cover solution. Now, however, a solution $\mathbf{x}$ of LP can't simply be ported back to a set cover solution. Indeed, we may get that in the solution to LP, the value of $\mathbf{x}_1 = 0.31$. What does this even mean? Should we pick S_1 in our solution, or not? At the end, our algorithm must make this choice. And now, having done almost a full course in randomized algorithms, you probably know the answer : $\mathbf{x}_1 = 0.31$ should probably be interpreted as "pick S_1 with probability 31%". Indeed, our final solution will be something like this, and this style of algorithm is called **randomized rounding**.

- **Randomized Rounding.**

1: **procedure** RANDOMIZED-ROUNDING($\mathbf{x} \in [0,1]^m$):
2: Select every $j \in [m]$ into A independently with probability $p_j := \min(1, 2 \ln n \cdot \mathbf{x}_j)$.

Theorem 2. The expected cost of the solution returned by randomized rounding is $\mathbf{Exp}[\mathsf{cost}(A)] \leq 2 \ln n \cdot \mathrm{LP}$, and with probability $\geq 1 - \frac{1}{n}$, it is a valid set cover.

Proof. The cost analysis is simply via linearity of expectation. Let Z_j be the indicator event that $j \in A$. Note, $\mathbf{Exp}[Z_j] = \mathbf{Pr}[Z_j = 1] \leq 2 \ln n \cdot \mathbf{x}_j$. Then,

$$\mathbf{Exp}[\mathsf{cost}(A)] = \mathbf{Exp}[\sum_{j=1}^{m} c_j Z_j] = \sum_{j=1}^{m} c_j \, \mathbf{Exp}[Z_j] \leq 2 \ln n \sum_{j=1}^{n} c_j \mathbf{x}_j \leq 2 \ln n \cdot \mathrm{LP}$$

Let us look at the event that A is **not** a set cover. Indeed, for any element $e \in U$, let $\mathcal{B}_e$ be the event e is not covered by the sets indexed by A. We want to upper bound $\mathbf{Pr}[\cup_{e \in U} \mathcal{B}_e] \leq \sum_{e \in U} \mathbf{Pr}[\mathcal{B}_e]$, by the union bound. To see an upper bound on the probability of $\mathcal{B}_e$, note

$$\begin{aligned}
\mathbf{Pr}[\mathcal{B}_e] &= \prod_{j : e \in S_j} \mathbf{Pr}[j \notin A] \\
&= \prod_{j : e \in S_j} (1 - p_j) \leq \prod_{j : e \in S_j} (1 - 2 \ln n \mathbf{x}_j) \\
&\leq \prod_{j : e \in S_j} e^{-2 \ln n \cdot \mathbf{x}_j} = e^{-2 \ln n \cdot \sum_{j : e \in S_j} \mathbf{x}_j} \qquad (1)
\end{aligned}$$

Now, we use the fact that the $\mathbf{x}_j$'s satisfy $\sum_{j : e \in S_j} \mathbf{x}_j \geq 1$, which plugging into (1) gives $\mathbf{Pr}[\mathcal{B}_e] \leq e^{-2 \ln n} = \frac{1}{n^2}$. Therefore, $\mathbf{Pr}[\mathcal{B}] \leq \sum_{e \in U} \mathbf{Pr}[\mathcal{B}_e] \leq \frac{1}{n}$. □