Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Randomization for Approximation

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT: given an instance with m clauses each containing exactly 3 distinct literals, find the assignment that satisfies as many of them as possible. Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies c clauses, your algorithm should produce an assignment that satisfies at least $\frac{7c}{8}$ clauses in expectation.

Hint: use linearity of expectation!

(b) Given a Max 3-SAT instance I, let OPT_I denote the maximum fraction of clauses in I satisfied by any variable assignment. What is the smallest value of OPT_I over all instances I? In other words, what is $\min_I \mathsf{OPT}_I$?

Hint: use part (a), and note that a random variable must sometimes be at least its mean.

Solution:

(a) Consider randomly assigning each variable a value. Let X_i be a random variable that is 1 if clause i is satisfied and 0 otherwise. We can see that the expectation of X_i is $\frac{7}{8}$. Note that $\sum X_i$ is the total number of satisfied clauses. By linearity of expectation, the expected number of clauses satisfied is $\frac{7}{8}$ times the total number of clauses. Since the optimal number of satisfied clauses is at most the total number of satisfied clauses, a random assignment will in expectation have value at least $\frac{7c}{8}$.

(b) Our randomized algorithm satisfies fraction 7/8 of clauses in expectation for any instance. So any instance must have a solution that satisfies at least fraction 7/8 of clauses (a random variable must sometimes be at least its mean).

This lower bound is tight: Consider an instance with 3 variables and all 8 possible clauses including these variables. Then any solution satisfies exactly 7 clauses.

2 MAX Mayhem

The Max-Cut problem (analogous to Min-Cut) tries to find the maximum sum of edge weights crossing a cut of the input graph. Remember that a cut must partition all vertices into exactly 2 disjoint subsets. There are no special s and t vertices.

The Max-Cut problem is NP-Hard, in contrast to Min-Cut, which can be solved in polynomial time. Our goal is to find a good approximation algorithm for Max-Cut.

(a) Explain why we cannot simply solve the equal-weighted version of the Max-Cut problem on G by inverting the edges of G (call this $\overline{G}$) and then solving Min-Cut on $\overline{G}$? Note that inverting edges means placing edges between pairs that don't have edges in G, but not between pairs that do have edges in G.

Solution: Note that the total number of pairs of vertices over cut S, T of graph G is $|S| \cdot |T|$. In other words, it is not constant over all cuts. So, even if MIN-CUT returns S and T in $\overline{G}$, this may not correspond to a very large cut in the original G if $|S|$ or $|T|$ is small!

(b) MAX-CUT has several 2-approximation algorithms; try to come up with one. *Hint: Perhaps think greedy or randomized!*

Note: In 1994, Goemans and Williamson came up with an algorithm with an approximation ratio of $\simeq 1.138$. This is currently believed to be the best possible efficient approximation. In fact, it turns out that any better *approximation* algorithm to the MAX-CUT problem is believed to be NP-hard!

Solution: Greedy solution: Define 2 bins, L and R, and arbitrarily assign the first vertex to one of these bins. For each subsequent vertex, assign it to the bin that maximizes the sum of crossing edge weights given the current partition.

Proof of approximation ratio: Let $v_1, \ldots, v_n$ be the order in which vertices are added. Let $\delta(v_i) = \{(v_i, w) \in E : w \in \{v_1, \ldots, v_{i-1}\}\}$ (edges between v_i and its neighbors which have already been added).

$$\sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{e \in \delta(v)} w(e)$$

Note that this inequality applies to individual vertices, so we sum over possible v.

$$w(C) = \sum_{v \in V} \sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{v \in V} \sum_{e \in \delta(v)} w(e)$$
$$= \frac{1}{2} \sum_{e \in E} w(e) \geq \frac{1}{2} \text{OPT}$$

Randomized solution: Assign each vertex to one of two subsets with uniform probability. Continuously repeat until at least half of the edges cross the cut.

Proof of approximation ratio: The expected weight of the randomized cut is:

$$w(C) = \sum_{e \in C} w(e) = \sum_{e \in E} w(e) P(e \in C)$$
$$= \frac{1}{2} \sum_{e \in E} w(e) \geq \frac{1}{2} \text{OPT}$$

(c) To help us evaluate how tight of a bound our approximation ratios give us, we introduce gap problems. We define gap_α-MAX-3SAT to be a decision problem that, upon inputs ϕ and m, returns YES if there is an assignment satisfying $\geq m$ clauses of ϕ, and NO if every assignment satisfies $\leq \alpha m$ clauses of ϕ. If neither statement is true, then it can output anything.

It turns out that the gap_α-MAX-3SAT problem is closely related to $\frac{1}{\alpha}$-approximations of MAX-3SAT. Explain how, if we have a $\frac{1}{\alpha}$-approximation algorithm, A, we can solve the gap_α-MAX-3SAT problem.

Challenge: you can also use a search version of the gap_α-MAX-3SAT problem to generate a $\frac{1}{\alpha}$ approximation. Try coming up with such an approximation algorithm!

Solution: Assume MAX-3SAT has a $\frac{1}{\alpha}$-approximation algorithm called A. Then we can define algorithm B that solves gap_α-MAX-3SAT below:

    (a) Let $m' = A(\phi)$

    (b) Return YES iff $m' \geq \alpha m$

    (c) Return anything otherwise

Assume algorithm B solves gap_α-MAX-3SAT below. Then we can find A, a $\frac{1}{\alpha}$-approximation algorithm for MAX-3SAT. Let n be the number of clauses of ϕ.

    (a) Run B on $(\phi, 1), \ldots, (\phi, n)$

    (b) Let m be the largest number of clauses (ϕ, m) for which B returned YES

    (c) Return αm

3 Randomized algorithms basics

Philosophy of analyzing randomized algorithms. The first step is to always identify a *bad event* – you want to identify when your randomness makes your algorithm fail. We will review some techniques from class using the following problem as a way to learn how to use them in the context of randomized algorithms.

> Let G be a bipartite graph with n left vertices, and n right vertices on $n^2 - n + 1$ edges.
> - Prove that G always has a perfect matching.
> - Give a polynomial in n time algorithm to find this perfect matching.

We will now use some common techniques to analyze the following algorithm `BlindMatching`:

- Let $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$ be independent and uniformly random permutations of $[n]$.
- If $\{\boldsymbol{\pi}(1), \boldsymbol{\sigma}(1)\}, \{\boldsymbol{\pi}(2), \boldsymbol{\sigma}(2)\}, \ldots, \{\boldsymbol{\pi}(n), \boldsymbol{\sigma}(n)\}$ is a valid matching output it.
- Else output `failed`.

Union Bound. Suppose $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are (not necessarily independent) Bernoulli random variables (i.e. random variables valued $\{0, 1\}$). Then, we have the following identity:

$$\Pr[\boldsymbol{X}_1 + \cdots + \boldsymbol{X}_n \geq 1] \leq \Pr[\boldsymbol{X}_1 = 1] + \Pr[\boldsymbol{X}_2 = 1] + \cdots + \Pr[\boldsymbol{X}_n = 1].$$

Now, using union bound, we analyze our algorithm for `BlindMatching`. Note that an output

$$M = (\{\boldsymbol{\pi}(1), \boldsymbol{\sigma}(1)\}, \ldots, \{\boldsymbol{\pi}(n), \boldsymbol{\sigma}(n)\})$$

is a valid perfect matching exactly when all edges of the form $\{\boldsymbol{\pi}(i), \boldsymbol{\sigma}(i)\}$ are present in G. A "bad event" happens if any of those pairs are not edges in G.

Let $\boldsymbol{X}_i$ be the indicator of the event that $\{\boldsymbol{\pi}(i), \boldsymbol{\sigma}(i)\}$ is *not* present in our graph.

1. What is the probability that $\boldsymbol{X}_i = 1$?

   Solution: The probability that a random (u, v) pair is not an edge where u is a left vertex and v is a right vertex is $\frac{1}{n} - \frac{1}{n^2}$.

2. Use the union bound to upper bound the probability that M is *not* a valid perfect matching.

   Solution: By union bounding over all n edges chosen, the probability that M is not a perfect matching is at most $1 - \frac{1}{n}$.

3. Conclude that G has a valid perfect matching.

   Solution: The previous part implies that M has at least $\frac{1}{n}$ probability of being a perfect matching, which means a perfect matching exists in G.

The upper bound obtained on the probability of our bad event, i.e. of M not being a valid perfect matching, is fairly high. In light of this, we introduce the technique of *amplification*.

This content is protected and may not be shared, uploaded, or distributed.

Amplification. The philosophy of amplification is that if we have a randomized algorithm that fails with probability p, we can repeat the algorithm many times and aggregate the output of all the runs to produce a new output such that the failure probability of the randomized algorithm is significantly smaller. Now consider the following algorithm `SpamBlindMatching`:

- Run `BlindMatching` independently T times.

- If at least one of the runs outputted a valid perfect matching, return the output of such a run.

- Else output `failed`.

To see the effectiveness of amplification, let us answer the following questions:

1. What is tight upper bound on the failure probability of `SpamBlindMatching`?

   Solution: The failure probability of `SpamBlindMatching` is bounded by $\left(1 - \frac{1}{n}\right)^T$.

2. How large should we set T if we want a failure probability of δ?

   Solution: Setting $T = n \ln(1/\delta)$ works because $\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{e}$.

Notice that the failure probability of `SpamBlindMatching` is not only lower than that of `BlindMatching`, but it can also be adjusted for based on the number of "amplifications" we make!

Now we switch gears and turn our attention to concentration phenomena and its usefulness in analyzing randomized algorithms.

Markov's inequality. Let $\boldsymbol{X}$ be a *nonnegative valued* random variable, then for every $t \geq 0$:

$$\Pr[\boldsymbol{X} \geq t] \leq \frac{\mathbf{E}[\boldsymbol{X}]}{t}.$$

1. Markov's inequality is *false* for random variables that can take on negative values! Give an example.

   Solution: Uniform ± 1 has expected value 0 but half chance of exceeding 0.

2. Give a tight example for Markov's inequality. In particular, given μ and t, construct a random variable $\boldsymbol{X}$ such that $\mu = \mathbf{E}[\boldsymbol{X}]$ and $\Pr[\boldsymbol{X} \geq t] = \frac{\mu}{t}$.

   Solution: Consider the random variable that is t with probability μ/t and 0 with probability $1 - \mu/t$.

Chebyshev's inequality. Let X be any random variable with well-defined variance[1], then

$$\Pr\left[|X - \mathbf{E}[X]| > t\sqrt{\mathbf{Var}[X]}\right] \le \frac{1}{t^2}.$$

To see the above inequality in action, consider the following problem: Let B be a bag with n balls, k of which are red and $n - k$ of which are blue. We do not have knowledge of k and wish to estimate k from ℓ independent samples (with replacement) drawn from B. Let X be the number of red balls sampled.

1. What is $\mathbf{E}[X]$?

   Solution: Defining X_i as the random variable that the i-th sample is red, we use linearity of expectation to yield
   $$\mathbf{E}[X] = \mathbf{E}[X_1 + \cdots + X_\ell] = \mathbf{E}[X_1]\ell = \frac{k}{n}\ell.$$

2. What is $\mathbf{Var}[X]$?

   Solution: Defining the same indicators as those in the previous part, we use independence of the X_i to get

   $$\mathbf{Var}[X] = \mathbf{Var}[X_1 + \cdots + X_\ell] = \mathbf{Var}[X_1] + \cdots + \mathbf{Var}[X_\ell] = \ell\frac{k}{n}\left(1 - \frac{k}{n}\right).$$

3. Choose a value for ℓ and give an algorithm that takes in n and X and outputs a number $\widetilde{k}$ such that $\widetilde{k} \in [k - \varepsilon\sqrt{k}, k + \varepsilon\sqrt{k}]$ with probability at least $1 - \delta$.

   Solution: The algorithm is to output $\frac{n}{\ell}X$ (given inputs n and X). $\mathbf{E}\left[\frac{n}{\ell}X\right] = k$ and $\mathbf{Var}\left[\frac{n}{\ell}X\right] = \frac{kn}{\ell}\left(1 - \frac{k}{n}\right) \le \frac{kn}{\ell}$. This quantity deviates from k by $\frac{1}{\sqrt{\delta}}\sqrt{\frac{kn}{\ell}}$ with probability at most δ. We wish to choose ℓ so that $\frac{1}{\sqrt{\delta}}\sqrt{\frac{kn}{\ell}} < \varepsilon$. This happens when $\ell = \frac{n}{\varepsilon^2\delta}$.

---

[1]In this course, all random variables will have well-defined variance (i.e. $\mathbf{Var}[\cdot] < \infty$).

4   4-cycles

We use $G(n, p)$ to denote the distribution of graphs obtained by taking n vertices and for each pair of vertices i, j placing edge $\{i, j\}$ independently with probability p.

(a) Compute the expected number of edges in $G(n, p)$? **Solution:** By linearity of expectation, $p\binom{n}{2}$ since there are $\binom{n}{2}$ potential edges in the graph.

(b) Compute the expected number of 4-cycles in $G(n, p)$? **Solution:** Each individual 4-cycle has probability p^4 of appearing, since it consists of 4 edges. There are 3 possible 4-cycles on any particular set of 4 vertices, and there are $\binom{n}{4}$ sets of 4 vertices in the graph. So, by linearity of expectation, the answer is $(p^4)(3)\binom{n}{4} = p^4 \cdot \frac{n(n-1)(n-2)(n-3)}{8}$.

(c) Describe, with proof of correctness, a polynomial time randomized algorithm that takes in n as input and in $\mathrm{poly}(n)$-time outputs a graph G such that G has no 4-cycles and the expected number of edges in G is $\Omega(n^{4/3})$. **Solution:** Let $\boldsymbol{G} \sim G(n, p)$ for $p = Cn^{-2/3}$; let $\boldsymbol{e}$ be the number of edges and let $\boldsymbol{c_4}$ be the number of 4-cycles in $\boldsymbol{G}$. Let $\boldsymbol{G'}$ be the graph obtained by deleting an edge from every 4-cycle, and output it; this deletion removes at most $\boldsymbol{c_4}$ edges, so the number of edges $\boldsymbol{e'}$ remaining in the graph satisfies: $\boldsymbol{e'} \geq \boldsymbol{e} - \boldsymbol{c_4}$, which means $\mathbf{E}[\boldsymbol{e'}] \geq \mathbf{E}[\boldsymbol{e}] - \mathbf{E}[\boldsymbol{c_4}] = \frac{pn(n-1)}{2} - \frac{p^4 n(n-1)(n-2)(n-3)}{4} \geq \frac{pn^2 - p^4 n^4}{4} = \frac{Cn^{4/3} - C^4 n^{4/3}}{4}$. Choosing C as, say, .5 finishes the proof, and each step takes polynomial time.