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.

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.

(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!

(c) To help us evaluate how tight of a bound our approximation ratios give us, we introduce gap problems. We define gap$_\alpha$-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$_\alpha$-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$_\alpha$-Max-3SAT problem.

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

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 $\boldsymbol{X}_i = 1$?

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

3. Conclude that G has a valid perfect matching.

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*.

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`?

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

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.

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 $\Pr[\boldsymbol{X} \geq t] = \frac{\mu}{t}$.

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

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

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 $\boldsymbol{X}$ be the number of red balls sampled.

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

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

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

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

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 $G(n,p)$?

(b) Compute the expected number of 4-cycles in $G(n,p)$?

(c) Describe, with proof of correctness, a polynomial time randomized algorithm that takes in n as input and in poly(n)-time outputs a graph G such that G has no 4-cycles and the expected number of edges in G is $\Omega(n^{4/3})$.

