DS 320 Algorithms for Data Science **Lecture #24 Worksheet**
Spring 2025 Prof. Kira Goldner and Jeffrey Considine

Approximation Algorithms: Randomized and Online

So far, we've been sure to always analyze runtime and often space as well. Another thing that we can add to our tradeoff is *optimality*.

Definition 1 (Approximation guarantee)**.** We say that an algorithm obtains an α-*approximation* to a maximization problem if in the worst case, the algorithm obtains at least $\textsc{Alg} \geq \alpha \, \textsc{opt}$ for $\alpha \in (0, 1)$.

Similarly, for a minimization problem, an algorithm obtains an α-approximation for $\alpha > 1$ if the algorithm's cost is at most $\textsc{Alg} \leq \alpha \, \textsc{opt}$.

Many times, it's useful to use randomness in our algorithm—to beat an adversary, or just because random choices do a good job of handling all of the different instances out there with some probability each. We give our guarantees in *expectation* over the randomness in the algorithm, i.e., for a maximization problem, we might say

$$\mathbb{E}_p[\textsc{Alg}] \geq \alpha \, \textsc{opt}$$

where p represents the random choices in the algorithm.

Sometimes, it is our *input* instance I that is random, in which case, we compare how both our algorithm do and how $\textsc{opt}$does in expectation over the random input that arrives, so we might prove a guarantee like

$$\mathbb{E}_I[\textsc{Alg}] \geq \alpha \, \mathbb{E}_I[\textsc{opt}].$$

Two probability essential facts to recall:

- Linearity of expectation: For any random variables X, x_i and constants c_i such that $X = \sum_i c_i x_i$, *regardless of whether or not the x_i's are independent*, $\mathbb{E}[X] = \sum_i \mathbb{E}[x_i]$. (This is not true for things like variance.)
- For a boolean (0/1) random variable x that is 1 with probability p, the expectation of x is $\mathbb{E}[x] = p$.

Randomized Algorithms

MAX SAT

Recall the definition of the 3-SAT problem: given a logical formula of n *boolean variables* $x_1, \ldots, x_n$ in *conjunctive normal form* (CNF), that is,

$$\phi = C_1 \wedge \cdots \wedge C_m$$

where C_i is one of m *clauses* each consisting of 3 disjoint *literals*, which are variables in either their positive or negative form, for example, perhaps

$$C_1 = (x_1 \vee x_2 \vee \overline{x}_3).$$

The goal of 3-SAT is to determine whether the formula ϕ can be satisfied.

Today, we look at the MAX SAT problem: very similar to 3-SAT, except that (1) each clause C_i may consist of any number of literals, (2) each clause i cause a weight w_i, and (3) rather than satisfy ϕ, our objective is to satisfy the maximum weight of clauses $\sum_i w_i C_i$.

For our algorithm, we'll try the *simplest possible idea*: set each boolean variable to true or false with equal probability. This is the most basic idea in randomized algorithms, and often times, it's actually enough, as we'll see today! In linear programs, we often had a $\{0, 1\}$ decision variable—true or false, take or don't take into a set, something like this. For these variables, the idea of "set to 1 with probability $\frac{1}{2}$" (and thus to 0 with equal probability) often works!

Randomized algorithm for MAX SAT:

Approximation Ratio:

Proof.

MAX CUT

In the maximum cut problem (MAX CUT), the input is an undirected graph $G = (V, E)$, along with a nonnegative weight $w_{ij} \geq 0$ for each edge $(i, j) \in E$. The goal is to partition the vertex set into two parts, U and $W = V \setminus U$, so as to maximize the weight of the edges whose two endpoints are in different parts, one in U and one in W. We say that an edge with endpoints in both U and W is *in the cut*. (In the case $w_{ij} = 1$ for each edge $(i, j) \in E$, we have an *unweighted* MAX CUT problem.)

Randomized algorithm for MAX CUT:

Approximation Ratio:

Proof.

Online Algorithms

The Ski Rental Problem

You're picking up a new hobby of skiing—you think. Every time that you rent skis, It costs \$100 and B-hundred dollars to buy skis (that is, we can think in units of hundreds of dollars). Maybe it'll be a lifelong hobby and you should invest that B up front, or maybe it's better to pay 1 each time in case it doesn't stick. Given that you don't know how many times you'll go skiing (say, some adversary decides whether you like it or not based on whether you purchase or not—isn't that how it always feels?), how should you decide whether to buy or not so that you can give *some guarantee?*

Definition 2 (Competitive ratio)**.** In an online setting, for a maximization problem, we say the *competitive ratio* of an algorithm is α if the algorithm obtains at least α-fraction of the *offline* full-information optimum OPT, that is,

$$\textsc{Competitive Ratio} = \frac{\textsc{Alg}}{\textsc{opt}} \geq \alpha.$$

And for a minimization problem, the algorithm obtains at most α-times the offline OPT: $\textsc{Alg} \leq \alpha \, \textsc{opt} \implies$:

$$\textsc{Competitive Ratio} = \frac{\textsc{Alg}}{\textsc{opt}} \leq \alpha.$$

Essentially, the *competitive ratio* is just the *approximation guarantee* in an online setting.

Online algorithm for Ski Rental:

Competitive Ratio:

Proof.

Lower bound proof: