CS 577 **Assignment 13 – Randomization** Spring 2023

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Kayley Seow Wisc id: kseow

Randomization

1. *Kleinberg, Jon. Algorithm Design (p. 782, q. 1).*

3-Coloring is a yes/no question, but we can phrase it as an optimization problem as follows. Suppose we are given a graph $G = (V, E)$, and we want to color each node with one of three colors, even if we aren't necessarily able to give different colors to every pair of adjacent nodes. Rather, we say that an edge (u, v) is *satisfied* if the colors assigned to u and v are different. Consider a 3-coloring that maximizes the number of satisfied edges, and let c^* denote this number. Give a polynomial-time algorithm that produces a 3-coloring that satisfies at least $\frac{2}{3}c^*$ edges. If you want, your algorithm can be randomized; in this case, the expected number of edges it satisfies should be at least $\frac{2}{3}c^*$.

We know that this will be a maximization problem, so we need an upper bound of c^*. Here we limit it to being less than equal to our number of edges, m. With three random colors, we have the probability $1/3$ where we have X_e, a edit can be a binary state of either 1 or 0. There are 9 different ways to color an edge, with three of them not satisfying, leaving us with a probability of $2/3$. with the random variable denoting satisfied edges by our linearity of expectations, we see that $\frac{2}{3}m$ is greater than or equal to $\frac{2}{3}c^*$.

2. *Kleinberg, Jon. Algorithm Design (p. 787, q. 7).*

In lecture, we designed an approximation algorithm to within a factor of 7/8 for the MAX 3-SAT Problem, where we assumed that each clause has terms associated with three different variables. In this problem, we will consider the analogous MAX SAT Problem: Given a set of clauses $C_1, \cdots, C_k$ over a set of variables $X = \{x_1, \cdots, x_n\}$, find a truth assignment satisfying as many of the clauses as possible. Each clause has at least one term in it, and all the variables in a single clause are distinct, but otherwise we do not make any assumptions on the length of the clauses: There may be clauses that have a lot of variables, and others may have just a single variable.

(a) First consider the randomized approximation algorithm we used for MAX 3-SAT, setting each variable independently to true or false with probability 1/2 each. Show that in the MAX SAT, the expected number of clauses satisfied by this random assignment is at least $k/2$, that is, at least half of the clauses are satisfied in expectation.

Let C be a clause with m literals, where each literal is either a variable or the negation of it. The probability of C satisfied by the random assignment is the probability that at least one is true. Because there is ½ chance of each. the probability that C is satisfied is $1-(\frac{1}{2})^m$. Let S be set of clauses that are satisfied, then, the expected # of satisfied is $E[|S|] = \Sigma_i$ probability (C_i satisfied) $\geq \Sigma_i (1-(\frac{1}{2})^m)$. The function $1-(\frac{1}{2})^k$ is non increasing function of k, M be max length of clause in set, we have $\sum_i (1-(\frac{1}{2})^{m_i}) \geq k(1-(1/2)^m)$. inequality follows that there are k clauses and each has at most M literals, $(\frac{1}{2})^m$ at most $\frac{1}{2}$, we have $\sum_i (1-(\frac{1}{2})^{m_i}) \geq k/2$, expected number of clauses satisfied at random is at least $\frac{k}{2}$ in MAXSAT problem.

(b) Give an example to show that there are MAX SAT instances such that no assignment satisfies more than half of the clauses.

C1: x_2
C2: $\neg x_2$
C3: x_2
C4: $\neg x_2$
C5: x_3
C6: $\neg x_3$

we have six clauses, and three variables. nomatter the truth assignment, we can satisfy at most three, out of the six. there is no possible assignment that satisfies more than half of the clauses.

(c) If we have a clause that consists only of a single term (e.g., a clause consisting just of x_1, or just of $\overline{x_2}$), then there is only a single way to satisfy it: We need to set the corresponding variable in the appropriate way. If we have two clauses such that one consists of just the term x_i, and the other consists of just the negated term $\overline{x_i}$, then this is a pretty direct contradiction. Assume that our instance has no such pair of "conflicting clauses"; that is, for no variable x_i do we have both a clause $C = \{x_i\}$ and a clause $C' = \{\overline{x_i}\}$. Modify the randomized procedure above to improve the approximation factor from 1/2 to at least 0.6. That is, change the algorithm so that the expected number of clauses satisfied by the process is at least $0.6k$.

Our new algorithm would be changed to identify clauses that consist of a single term, and set the corresponding variable to satisfy the clause, ensuring we satisfy as many clauses as possible. If we have no such clauses, then we proceed with random assignment, but expected number of clauses is modified in this procedure to be at least 0.6k.

Pseudocode:

Set each variable independently to T or F, ½ probability each

For each variable x_i, if there exists clause $C = \{x_i\}$, set x_i to true,

For each variable x_i, there exists clause $C = \{\neg x_i\}$, set x_i to false.

At most 40% of the clauses are containing 2 or more variables, and probability that they will be assigned a clause is at most $(\frac{1}{2})^k$, k being number of variables in a clause. The expected number of clauses satisfied is at most $(\frac{1}{2})^k$ the number of clauses, and the expected number of clauses is at most 0.4k. For each of the remaining variables, we set the corresponding variable to satisfy the probability min ½, and expected number is 0.5 times the number of these clauses.

Combining these, expected number of clauses satisfied procedure is at least $0.5k + 0.6(0.4k) = 0.6k$.

(d) Give a randomized polynomial-time algorithm for the general MAX SAT Problem, so that the expected number of clauses satisfied by the algorithm is at least a 0.6 fraction of the maximum possible. (Note that, by the example in part (a), there are instances where one cannot satisfy more than $k/2$ clauses; the point here is that we'd still like an efficient algorithm that, in expectation, can satisfy a 0.6 fraction of the maximum that can be satisfied by an optimal assignment.)

A key modification to this algorithm that makes it different is to choose an unsatisfied clause C with probability proportional to the number of variables in C, because larger clauses have greater impact on satisfaction of the formula, so we want to focus on these.

pseudocode

- Choose random assignment of T or F to x_i with probability 1/2
- while (unsatisfied clauses exist):
  - → choose unsat. clause C with probability proportional to number of variables in C
  - → choose variable x_i uniformly at random, from variables in C
  - → flip assignment of x_i to its negation

(Flips the assignment until all are satisfied.)

We claim algo satisfies at least 0.6 · Let OPT be max number of clauses satisfied by assignment, p be prob. clause satisfied, and expected from random is kp. When we flip the clauses, the probability of satisfying clauses is $\frac{1}{2}m$, and probability is $(\frac{1}{2}m) \times (\frac{1}{m})$ because we have to choose the next variable. Expected number of clauses is now atleast $(\frac{1}{2m^2})$. We need to prove that the Σ # of iterations until all clauses are satisfied, T, that the algorithm is $kpt^T/2m^2$, is at least 0.6(OPT). We can show that p is atleast $\frac{1}{2}$, so kp is at least $k/2$, For expected value of T, we satisfy at least one new clause, with probability at least $\frac{1}{2pm^2}$, so expected number of iterations is at most $2m^2(OPT)$. expected value of $/2m^2$ is at most OPT. Expected number of clauses satisfied by algorithm is at least $\frac{k}{2} + OPT$, which is at least 0.6(OPT) since OPT is at least $k/2$, therefore satisfying at least 0.6 fraction of max possible clauses.

3. *Kleinberg, Jon. Algorithm Design (p. 789, q. 10).*

Consider a very simple online auction system that works as follows. There are n *bidding agents*; agent i has a bid b_i, which is a positive natural number. We will assume that all bids b_i are distinct from one another. The bidding agents appear in an order chosen uniformly at random, each proposes its bid b_i in turn, and at all times the system maintains a variable b^* equal to the highest bid seen so far. (Initially b^* is set to 0.) What is the expected number of times that b^* is updated when this process is executed, as a function of the parameters in the problem?

We will record b^* as X, a random variable equal to b^*, and it is equal to $X_1 + X_2 + X_3 + \ldots + X_{n-1} + X_n$, where X_i is equal to 1 if it causes the b^* to be updated, and $X_i = 0$ otherwise. In the case that $X_i = 1$, iff in this case, the largest one will be at the end. But, due to how the problem is set up, the bid could be anywhere, giving us the probability of $EX_i = 1/i$. The number of permutations in which the number at position i is larger than any of the numbers it comes after can be put as $\binom{n}{i}$, where the first i numbers in this amount of ways, putting the largest in position i, and ordering the remainder in $(i-1)$ ways, ordering the remaining $(n-i)$ numbers in $(n-i)!$ ways. we end up with $\binom{n}{i}(i-1)!(n-i)! = n!/i$, and dividing by $n!$, we end up with $EX_i = 1/i$. By the linearity of the expectation given by the problem, we end up with $EX = \sum_{i=1}^{n} EX_i = \sum_{i=1}^{n} 1/i = H_n = \Theta(\log(n))$.

4. Recall that in an undirected and unweighted graph $G = (V, E)$, a cut is a partition of the vertices $(S, V \backslash S)$ (where $S \subseteq V$). The size of a cut is the number of edges which cross the cut (the number of edges (u, v) such that $u \in S$ and $v \in V \backslash S$). In the MAXCUT problem, we try to find the cut which has the largest value. (The decision version of MAXCUT is NP-complete, but we will not prove that here.) Give a randomized algorithm to find a cut which, in expectation, has value at least 1/2 of the maximum value cut.

For this algorithm, we want to start with a random cut, and make it better by moving vertices between the partitions. We choose the portion that has more connections to the partition and move it opposite, increasing the size of the cut, repeating this k times, we increase the probability that it will be found, that is at least half the size of the min cut.

- Initialize cut by randomly selecting random subset S of vertices
- For k iterations:
  Compute the size of the cut, for partition (S, ~S)
  For each vertex in V
    calculate # of edges connecting v to vertices in S, number of edges connecting v to vertices in V\S, difference between values.
  Choose vertex v at random from V, probability proportional to d(v).
  If v is in S, we remove and otherwise, add to S
- Output the final cut of (S, V\S)

The runtime of this algorithm is O(kn) where n is the number of vertices and k is the number of iterations.

5. Implement an algorithm which, given a MAX 3-SAT instance, produces an assignment which satisfies at least 7/8 of the clauses, in either C, C++, C#, Java, or Python.

   The input will start with a positive integer n giving the number of variables, then a positive integer m giving the number of clauses, and then m lines describing each clause. The description of the clause will have three integers x y z, where $|x|$ encodes the variable number appearing in the first literal in the clause, the sign of x will be negative if and only if the literal is negated, and likewise for y and z to describe the two remaining literals in the clause. For example, 3 −1 −4 corresponds to the clause $x_3 \wedge \overline{x_1} \wedge \overline{x_4}$. A sample input is the following:

```
10
5
-1 -2 -5
6 9 4
-9 -7 -8
2 -7 10
-1 3 -6
```

   Your program should output an assignment which satisfies at least $\lfloor 7/8 \rfloor m$ clauses. Return n numbers in a line, using a ± 1 encoding for each variable (the ith number should be 1 if x_i is assigned TRUE, and -1 otherwise). The maximum possible number of satisfied clauses is 3, so your assignment should satisfy at least $\lfloor \frac{7}{8} \times 3 \rfloor = 2$ clauses. One possible correct output to the sample input would be:

```
-1 1 1 1 1 1 -1 1 1 1
```