CS 577 # Assignment 13 – Randomization Spring 2024

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: Ki Min Kang Wisc id: 908 404 5062

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 will use a randomized algorithm to maximize the number of edges between nodes of different colors.

1. For each vertex in the graph $G=(V,E)$, assign one of the three available colors uniformly at random.
2. The probability that an edge (u,v) is not satisfied is $\frac{1}{3}$ since there are three colors and we are choosing randomly.
3. Conversely, the probability that an edge (u,v) is $1-\frac{1}{3}=\frac{2}{3}$.
4. Let c^* be the maximum number of edges that could be satisfied by an optimal coloring. In expectation, the randomized algorithm satisfies at least $\frac{2}{3} \times c^*$ edges because each edge has a $\frac{2}{3}$ chance of satisfied
5. To achieve this, iterate through each edge of the graph once and for each edge, calculate the probability of it being satisfied by the current coloring. The sum of these probabilities over all edges gives us the expected number of satisfied edges.

The randomized algorithm runs in polynomial time because it iterates through the edges once and the probability calculation for each edge is done in constant time.

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.

We are given a set of clauses $C_1, C_2, \cdots, C_k$ over a set of boolean variable $X = \{x_1, x_2, \cdots, x_n\}$.
Each clause C_i is a disjunction of literals, and we wish to assign truth values to the variables such that as many clauses as possible are satisfied.
The randomized approximation algorithm assigns true or false to each variable x_i independently with a probability of 1/2 each
For any single clause C_i, the probability that it is not satisfied is $(1/2)^m$ where m is the number of literals in the clause. This is because each literal has a 1/2 chance of being false, and all literals would need to be false to not satisfy the clause
Therefore, the probability that the clause is satisfied is $1-(1/2)^m$.
Since $m \geq 1$ for any clause, the smallest this probability can be is $1 - 1/2 = 1/2$ for a single clause.

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

1. Consider a Max SAT instance with variables $x_1, x_2, \cdots, x_n$.
2. Construct clauses in pairs for each variable x_i such that one clause contains the variable x_i and the other contains its negation $\neg x_i$.
3. This means for each x_i, we have a clause (x_i) and a clause $(\neg x_i)$.
4. The construction leads to a total of $2n$ clauses if we have n variables.
5. No matter how we assign truth values to the variables, one clause in each pair will always be false. If x_i is true, then $(\neg x_i)$ will be false, and vice versa.
6. Therefore, the maximum number of clauses that can be satisfied is n, which is exactly half of the total number of clauses $2n$.

(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 $0.6k$.

When a clause consists of a single term (say x_i), we assign the variable a true value with a probability greater than 1/2 to increase the likelihood of satisfying these singleton clauses. Let's denote this probability as p. For singleton clauses, the probability of being satisfied is now p.

For clauses with multiple terms, the probability of not satisfying the clause is the product of the probabilities of all individual literals being false. For a clause with m variables, this would be $(1-p)^m$.

Therefore, the probability that a multi-variable clause is satisfied is $1-(1-p)^m$.

To maximize the number of expected satisfied clauses, we need to find the optimal probability p that maximizes the minimum expected satisfaction rate for any clause.

By setting $p = 1-p^2$, we solve for p which gives us $p = (\sqrt{5}-1)/2$ ensuring that the expected satisfaction rate for any clause is at least p.

With this choice of p, the expected number of satisfied clauses is at least $0.618k$, where k is the total number of clauses.

The result is an approximation factor improved from 0.5 to at least 0.6, achieving a better expected outcome for the randomized algorithm on the MAX SAT problem.

(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.)

- Assign truth values to variables in such a way that each variable is independently true with a probability p and false with a probability $1-p$, where p is value that maximizes the expected satisfaction of clauses, found to be $p = (\sqrt{5}-1)/2 \approx 0.618$.
- Conflicting clauses are pairs of clauses where one contains a single variable x_i and the other contains its negation $\neg x_i$. Since they can't both be true, this requires special handling.
- Before assigning values, go through the set of clauses and remove one clause from each conflicting pair. This step ensures there are no direct contradictions in the clauses to be satisfied.
- After removing the conflicting clauses, apply the randomized assignment from first step to the remaining clauses. For each clause C_i with m variables, the probability it is not satisfied is $(1-p)^m$, and hence the probability it is satisfied is $1-(1-p)^m$.
- Sum the probabilities of satisfaction across all clauses. The sum gives the expected number of satisfied clauses.
- The expected number of satisfied clauses post-preprocessing is at least $0.618k$, where k is the number of remaining clauses after preprocessing.
- The preprocessing step and the assignment of values are both achievable in polynomial time relative to the number of clauses and variables.

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 calculate the expected number of updates:

- Start by considering the first bid b_1. Since it's the first bid, it will automatically be the highest bid so far, and b^* will be updated once.
- The second bid b_2 has a 1/2 chance of being higher than the first bid since the order is random. Therefore, it has a 1/2 chance of updating b^*.
- The third bid b_3 will be the highest only if it is greater than both previous bids. Since the bids are distinct and ordered randomly, b_3 has a 1/3 chance of being the highest and thus updating b^*.
- Generalizing this, the i-th bid b_i, has a $1/i$ chance of being the highest bid out of the first i bids and updating b^*.
- The expected number of updates after all n bids is the sum of the individual probabilities for each bid being the highest when it is made:

  Expected number of updates $= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$
- This sum is known as the n-th Harmonic number. It does not have a simple closed form, but it grows logarithmically with n and can be approximated by $\ln(n) + \gamma$, where γ is the Euler-Mascheroni constant, approximately 0.577.

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.

1. Each vertex $v \in V$ is assigned to set S with probability $1/2$, independently from other vertices
2. For each edge $(u, v) \in E$, the probability that it crosses the cut is $1/2$. This is because there are two favorable outcomes out of the possible outcomes.
3. Because each edge has a $1/2$ chance of being in the cut, the expected number of edges in the cut for any given edge is $1/2$. Therefore, the expected size of the cut, which is the sum of these expectations for all edges is $E/2$, where E is the total number of edges in the graph.
4. Summing up over all edges, the expected size of the cut produced by this random assignment is $|E|/2$, which is half the total number of edges.
5. The maximal cut can't have more edges than E, so the expectation of $|E|/2$ means the randomized algorithm finds a cut that in expectation has at least half the number of edges of the maximum value cut.
6. The expected value of the cut produced by the randomized algorithm is at least $1/2$ of the optimal cut. This holds due to the linearity of expectation and the the independence of vertex assignments.

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, Python, or Rust.

   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 $x_3 \vee \overline{x_1} \vee \overline{x_4}$. 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 $\lfloor \frac{7}{8} m \rfloor$ 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 5, so your assignment should satisfy at least $\lfloor \frac{7}{8} \times 5 \rfloor = 4$ clauses. One possible correct output to the sample input would be:

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