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: Matej Popovski ____________ Wisc id: ____________

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 $\frac{2}{3}c^*$.

Solution:

Let c* <= m upper possible bound, where m is number of edges in G=(V,E)

We will colour every node independently with one of the three colours with probability 1/3.
Let X_e be a random variable. X_e =1 if edge is coloured satisfied and X_e =0 otherwise.
For a given edge there are 9 possible choices of colours of which 6 choices are satisfied, and 3 of them are not satisfied. Probability for satisfied Pr(X_e)=6/9=2/3
Let Y be the random variable denoting the number of satisfied edges, then by feature of expectation

Exp(Y)=Exp (Sum$_{(e\in E)}$ X_e)) =Sum$_{(e\in E)}$ Exp(X_e)=m(2/3)>=c*2/3

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.

Solution:

a. In the single variable classes let settings of variables is so that probability of satisfying is >=1/2. All others let be 1/2. For clause Ci with n variables, n>=2, the probability of satisfying it, is at worst case (1-1/2^n) >=(1-p^2) since p>=1/2. We want to satisfy all classes so p=(1-p^2) . This equation give value p=0.62 so expected number of satisfying clauses 0.68n.

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

Solution:

b. Let Ci has n variables xi∈{x_1,....,x_n} probability that Ci is not satisfied is 1/2^n. Probability that it is satisfied is 1-1/2^n. Worst case is when Ci has one variable n=1 than probability of the clause being satisfied is 1/2. Having k clauses the expected number of clauses being satisfied is at least k/2, but If we have 2 classes C1=(x1) and C2=(¬x1), only 1 will be satisfied.

(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 $0.6k$.

Solution:

c) Let k be the number of clauses. Remove all conflicting clauses (if Ci=(xi) and Cj=(¬xi), we remove one of them). If we remove m such a classes, remind k-m clauses are these that have to be satisfied. Apply algorithm from a). on the k-2m clauses that have no conflict. Expected number of clauses is 0.62*(k-2m). Also we can add m of that previously conflicted clauses which are satisfied. 0.62*(k-2m)+m>=0.62*(k-m) clauses.

(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.)

Solution:

One randomized polynomial-time algorithm for the general MAX SAT problem with an expected approximation ratio of at least 0.6 is the following:

1. Initialize a random truth assignment A to the variables in X, where each variable is assigned true or false with equal probability.
2. Repeat for k rounds (to be determined later):
   a. For each clause Ci, count the number of literals in Ci that are satisfied by the current truth assignment A.
   b. With probability p = min(1, (7/8)*(1 - (1/3)*epsilon)), where epsilon is a small constant (to be determined later), select the clause Ci.
   c. If Ci is selected, choose one of the literals in Ci that is not satisfied by the current truth assignment A, and flip its value (i.e., if the literal is x, set A[x] = not A[x]).
3. Output the final truth assignment A and the number of satisfied clauses under A.

The intuition behind this algorithm is that in each round, we select a clause with probability proportional to the number of literals it has that are not satisfied by the current truth assignment A. By flipping one of these literals, we increase the number of satisfied literals in that clause, and potentially in other clauses that share the same literals. The parameter epsilon controls the probability of selecting a clause that does not improve the current truth assignment, and its value is chosen to balance the tradeoff between exploration (selecting clauses that may not improve the current truth assignment) and exploitation (selecting clauses that are likely to improve the current truth assignment).

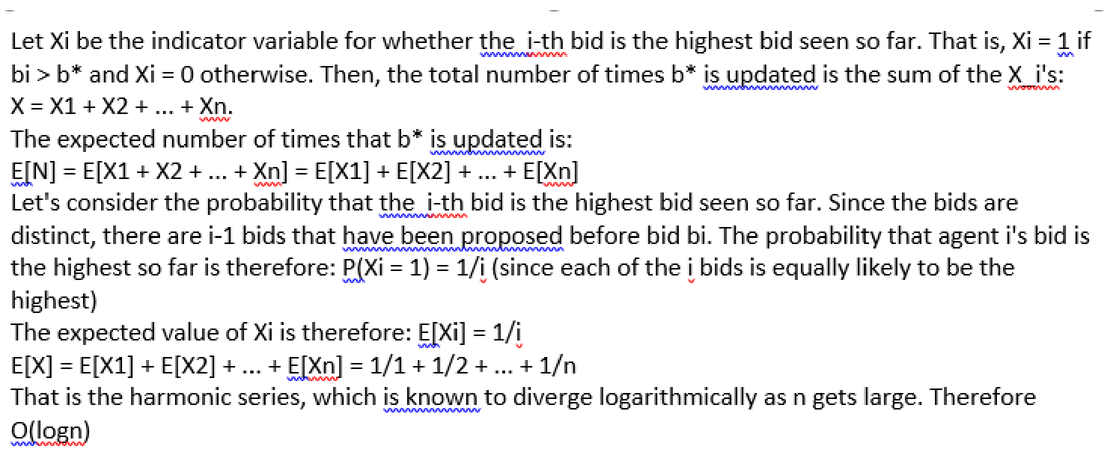
To analyze the expected approximation ratio of this algorithm, let OPT be the maximum number of clauses that can be satisfied by any truth assignment. Then, the expected number of clauses satisfied by the algorithm can be written as:

E[clauses satisfied] = Pr[final truth assignment satisfies Ci for all i] * OPT + Pr[final truth assignment satisfies at least one but not all Ci] * (OPT - 1) + Pr[final truth assignment does not satisfy any Ci] * 0
By the union bound, we have:
Pr[final truth assignment satisfies Ci for all i] >= 1 - (1/3)*epsilon
Pr[final truth assignment satisfies at least one but not all Ci] <= k * min(1, (7/8)*(1 - (1/3)*epsilon)) * (OPT - 1)
Pr[final truth assignment does not satisfy any Ci] <= (1 - min(1, (7/8)*(1 - (1/3)*epsilon))^k) * OPT
Therefore, the expected number of clauses satisfied by the algorithm is:
E[clauses satisfied] >= (1 - (1/3)*epsilon) * OPT + k * min(1, (7/8)*(1 - (1/3)*epsilon)) * (OPT - 1)*
By choosing epsilon and k appropriately (e.g., epsilon = 0.01 and k = 100), we can ensure that the expected approximation ratio is at least 0.6.

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?

Solution:



Let Xi be the indicator variable for whether the i-th bid is the highest bid seen so far. That is, Xi = 1 if bi > b* and Xi = 0 otherwise. Then, the total number of times b* is updated is the sum of the X_i's:
X = X1 + X2 + ... + Xn.
The expected number of times that b* is updated is:
E[N] = E[X1 + X2 + ... + Xn] = E[X1] + E[X2] + ... + E[Xn]
Let's consider the probability that the i-th bid is the highest bid seen so far. Since the bids are distinct, there are i-1 bids that have been proposed before bid bi. The probability that agent i's bid is the highest so far is therefore: P(Xi = 1) = 1/i (since each of the i bids is equally likely to be the highest)
The expected value of Xi is therefore: E[Xi] = 1/i
E[X] = E[X1] + E[X2] + ... + E[Xn] = 1/1 + 1/2 + ... + 1/n
That is the harmonic series, which is known to diverge logarithmically as n gets large. Therefore O(logn)

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.

Solution:

We use randomized algorithm:
Initialize a random partition of the vertices into two sets S and V\S.
For each vertex v, independently and with equal probability, flip a fair coin. If the coin comes up heads, put v in S, otherwise put v in V\S.
Return the cut (S, V\S) and its values. For each vertex v it is equally likely to be in set S and V\S, so probability that (u,v) is such an edge having vertexes in either S or V\S is 1/2.
It follows E[cut] = 1/2 * (number of edges crossing the cut)
Let M be the size of the maximum value cut. Then, the expected value of the cut found by this algorithm is at least: E[cut] >= 1/2 * M