CS 670 (Spring 2025) — Assignment 6

Due: 05/07/2025

This homework can be submitted until 11:30am on Wednesday, 05/07/2025, but not later, regardless of how many late days you have left. After 11:30am on 05/07/2025, you can pick up sample solutions in David's office.

(1) Describe at least one way in which what you have learned in this class helps you gain a new perspective on your own research area, or gives you new tools which you think will help improve your own research. You can talk about broad understanding, or about a specific problem for which you think you might now have better approaches available, or anything else relating to your own work. If you are not research-active, then talk about perspectives on another area of CS you are most interested in.

Your answer to this question should reflect some serious thought, not superficial comments that you come up with in 5 minutes. Also, please submit your answer to this question on a separate piece of paper, since David will "grade" it.

(2) [10 points]
Problem 13.6 from the textbook.

(3) [10 points]
Problem 13.15 from the textbook.

(4) [10 points]
Here, we explore another randomized approximation algorithm for Max-Sat. Different from "Johnson's Algorithm" from class, we will not assume that each clause has the same number $k = 3$ of literals. Now, each clause is an OR of one or more literals, each of which is a variable or negated variable.

Specifically, we will let C_j^+ and C_j^- be the indices of variables which occur in clause j unnegated (C_j^+) or negated (C_j^-). We can write an integer LP for this problem as follows, with $z_j \in \{0, 1\}$ capturing whether clause j is satisfied (1 meaning satisfied, 0 meaning not satisfied), and $y_i \in \{0, 1\}$ capturing if variable i is true (1) or false (0).

$$\begin{array}{lll} \text{Maximize} & \sum_{j=1}^{m} z_j & \\ \text{subject to} & z_j \leq \sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) & \text{for all } j \\ & y_i \geq 0 & \text{for all } i \\ & z_j \geq 0 & \text{for all } j \\ & y_i, z_j \in \{0, 1\} & \end{array}$$

As usual, by getting rid of the last (integrality) constraint, we relax the ILP to an LP.

Give a (polynomial-time) randomized rounding algorithm and prove that it gives a $(1 - 1/e)$ approximation. Notice that this is worse than 7/8 for Max-3-Sat, so it is mostly useful when we have lots of short clauses of one or two literals.

Hint: The inequality $1 - (1 - z/k)^k \geq (1 - (1 - 1/k)^k) \cdot z$ for all integers $k \geq 1$ and all $z \in [0, 1]$ is probably going to be very useful in your analysis. For full credit, if you are going to use it, you need to prove it, though. Plotting the two functions might give you an idea of how to prove it.

(5) [10 points]
Consider the problem of Binary Search when you might get incorrect answers. Our model is as follows: you have a sorted "array" of n elements. There is some element x you are looking for, which we assume is definitely somewhere in the array. When you compare x with position j of the array, there are three possibilities:

(a) If x is in fact in position j, you definitely are told so.

(b) If x is to the left of position j, then with probability $p > 1/2$, you are correctly told that it is to the left. But with probability $1 - p$, you are instead told that it is to the right.

(c) If x is to the right of position j, then with the same probability p, you are correctly told that it is to the right. But with probability $1 - p$, you are instead told that it is to the left.

You would like to find the element x with probability at least $1-\epsilon$ (where ϵ is a given input parameter), using few queries. Notice that simply running standard binary search will not work: if the first query tells you to go left, in standard binary search, all future queries will be to the left half, so if the answer was wrong, Binary Search can never recover. In particular, if $p < 1 - \epsilon$ (which it will usually be), you cannot achieve the desired $1 - \epsilon$ guarantee.

Consider the following way of fixing the algorithm: whenever you are querying a position j, repeat that query k times, and take a majority vote of the k answers to decide whether to go left or right. Reverse engineer a suitable k so that the algorithm overall succeeds with probability at least $1-\epsilon$, and give a complete analysis. The number of queries will presumably depend on p and ϵ, and that's ok. Also, the dependence of the number of queries on n will probably be a bit worse than $O(\log n)$, but it shouldn't be a whole lot worse.

(6) Chocolate Problem (1 chocolate bar): Derive a better algorithm for the previous problem (Binary Search with probabilistically incorrect answers), such that the dependence of the number of queries on n is now actually $O(\log n)$. We still want success with probability at least $1 - \epsilon$, and the number of queries may of course depend on ϵ and p.

Hint: Keep track of a weight for each array position j. Multiply this weight by p every time the position j agrees with the answer, and by $1 - p$ every time it disagrees. Then do something "smart" with the weights that generalizes the case $p = 1$ from standard Binary Search.