CS 170 Homework 12

Due **Friday 11/22/2024, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Upper Bounds on Algorithms for NP Problems

(a) Recall the 3-SAT problem: we have n variables x_i and m clauses, where each clause is the OR of at most three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses, or report that none exists.

Give a $O(2^n m)$-time algorithm for 3-SAT. Just the algorithm description is needed.

(b) Using the fact that 3-SAT is **NP**-hard, give a $O(2^{n^c})$-time algorithm for any problem in **NP**, where c is a constant (that can depend on the problem). Just the algorithm description and runtime analysis is needed.

*Hint: Since 3-SAT is **NP**-hard, you can use it to solve any problem in **NP**!*

Note: This result is known as **NP** $\subseteq$ **EXP**.

(c) Recall the halting problem from CS70: Given a program (as e.g. a `.py` file), determine if the program runs forever or eventually halts. Also recall that there is no finite-time algorithm for the halting problem. Let us define the input size for the halting problem to be the number of characters used to write the program.

Given an instance of 3-SAT with n variables and m clauses, we can write a size $O(n+m)$ program that halts if the instance is satisfiable and runs forever otherwise. So there is a polynomial-time reduction from 3-SAT to the halting problem.

Based on this reduction and part (b): Is the halting problem **NP**-hard? Is it **NP**-complete? Justify your answer.

Solution:

(a) Enumerate all 2^n assignments. We can check if each assignment is satisfying in $O(m)$ time. So we can find a satisfying assignment if one exists $O(2^n m)$-time.

(b) Any problem in NP can be reduced to an instance of 3-SAT in polynomial time. Since the reduction takes polynomial time, and it takes at least $O(1)$ time to write down a variable/clause, the 3-SAT instance can't have more than n^b variables and n^b clauses for some constant b (for all sufficiently large n). So our algorithm is to reduce to 3-SAT and then solve this 3-SAT instance in $O(2^{n^b} n^b) = O(2^{n^{b+1}})$ time using part a. This is $O(2^{n^c})$ for $c = b + 1$.

Alternate approach: Since the problem is in NP, if the answer to the problem is "yes", there is some polynomial-size "witness" to the problem that we can feed to a verification algorithm and have it output "yes" in polynomial time. If the answer is no, none of these witness cause it to output "yes".

So, we can enumerate over all witnesses, and feed them to this verification algorithm, and output yes if any run of the verification algorithm outputs yes. Since there are polynomially many witnesses, and the verification algorithm runs in polynomial time, this takes $O(2^{n^c})$ time for some constant c.

(c) The reduction implies the halting problem is NP-hard. We can show the halting problem isn't in NP, and thus it isn't NP-complete: By part (a), any problem in NP has a finite-time algorithm, but the halting problem doesn't have one. So the halting problem isn't NP-complete.

3 k-XOR

In the k-XOR problem, we are given n boolean variables $x_1, x_2, \ldots, x_n$, a list of m clauses each of which is the XOR of exactly k distinct variables (that is, the clause is true if and only if an odd number of the k variables in the clause are true), and an integer c. Our goal is to decide if there is some assignment of variables that satisfies at least c clauses.

(a) In the Max-Cut problem, we are given an undirected unweighted graph $G = (V, E)$ and integer α and want to find a cut $S \subseteq V$ such that at least α edges cross this cut (i.e. have exactly one endpoint in S). Give and argue correctness of a reduction from Max-Cut to 2-XOR.

Hint: every clause in 2-XOR is equivalent to an edge in Max-Cut.

(b) Give and argue correctness of a reduction from 3-XOR to 4-XOR.

Solution:

(a) We create a variable for each vertex i, and a clause x_i XOR x_j for each edge (i, j). We choose $\alpha = c$.

For any assignment, consider the cut such that S contains all vertices i for which x_i is true in this assignment. For each edge (i, j) crossing this cut, its corresponding clause is true because exactly one of x_i, x_j is true. For each edge not crossing this cut, its corresponding clause is false because either both x_i, x_j are true or neither is true. This proves correctness of the reduction.

(b) The reduction is to add a new variable z and add z to every clause in the Max 3-XOR instance and choose the same value of c to get a Max 4-XOR instance.

Given an assignment that satisfies c clauses in the Max 3-XOR instance, the same assignment plus z set to false satisfies c clauses in the Max 4-XOR instance. Given an assignment that satisfies c clauses in the Max 4-XOR instance, if z is false, then the same assignment minus z satisfies c clauses in the Max 3-XOR instance. Otherwise, z is true, and the same assignment minus z and negating all other variables satisfies c clauses in the Max 3-XOR instance.

To see this, consider any clause that was true in Max 4-XOR. Deleting z takes it from having an odd number of true variables to an even number. Then, since 3 is odd, negating all variables brings it back to having an odd number of true variables. Similarly, any clause that was false in the Max 4-XOR instance has an even number of true variables. Deleting z causes it to have an odd number of true variables, and then negating all other variables brings it back to even.

4 Dominating Set

A dominating set of a graph $G = (V, E)$ is a subset D of V, such that every vertex not in D is a neighbor of at least one vertex in D. Let the Minimum Dominating Set problem be the task of determining whether there is a dominating set of size $\leq k$. Show that the Minimum Dominating Set problem is NP-Complete. You may assume that G is connected.

Hint: Try reducing from Vertex Cover or Set Cover.

Solution:

Proof that Minimum Dominating Set is in NP.
Given a possible solution, we can check that it is a solution by iterating through the vertices not in the solution subset D and checking if it has a neighbor in D by iterating through the adjacent edges. This would take at most E time because you would at most check every edge twice. We should also check that the number of vertices in D is less than k, which would take at most V as k should be less than E. So, we can verify in $O(E)$ time which is polynomial.

Proof that Minimum Dominating Set is NP-Hard.

Reduction from Vertex Cover to Dominating Set
Minimum Vertex Cover is to find a vertex cover (a subset of the vertices) which is of size $\leq k$ where all edges have an endpoint in the vertex cover.

Suppose $G(V, E)$ is an instance of vertex cover and we are trying to find a vertex cover of size $\leq k$. For every edge (u, v), we will add a new vertex c and two edges (c, u), and (c, v). We will pass in the same k to the dominating set. This is a polynomial reduction because we are adding $|E|$ vertices and $2|E|$ edges.

Proof of Correctness of Reduction
1. **If minimum vertex cover has a solution, then minimum dominating set that corresponds to it has a solution.**
Suppose we have some minimum vertex cover of size k. By definition of vertex cover, every edge has either one or both endpoints in the vertex cover. Thus if we say that the vertex cover is a dominating set, every original vertex must either be in the dominating set or adjacent to it. Now we have to figure out whether the vertices we added for every edge are covered. Since every edge has to have one or both endpoint in the vertex cover, the added vertices must be adjacent to at least one vertex in the vertex cover. Thus the vertex cover maps directly to a dominating set in the transformed problem.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum vertex cover has a solution.**
Suppose we have some minimum dominating set of size k of the transformed format. Vertices in the dominating set either must come from the original vertices or the vertices we added. If they don't come from the vertices we added in the transformation, then from the logic stated in the previous direction, the dominating set corresponds directly to the vertex cover. If some vertices come from the transformation, then we can substitute the vertex for either

of the endpoints of the edge it corresponds to without changing the size of the dominating set. Thus, we can come up with a dominating set of size k that only uses vertices from our original problem, which will directly match to a minimum vertex cover of size k in the original problem.

Alternative Proof that Minimum Dominating Set is NP-Hard.

Reduction from Set Cover to Dominating Set

Minimum Set Cover is to find a set cover (a subset of all the sets) which is of size $\leq k$ where all elements are covered by at least one set of the set cover.

Suppose (S, U) is an instance of set cover where U denotes the set of all distinct elements and S is a set of subsets S_i of U. We will construct a graph $G = (V, E)$ as follows. For each element u in U construct a vertex; we will call these "element vertices". For each possible S_i construct a vertex; we will call these "set vertices". Connect each vertex S_i to all u in S_i.

Notice that if we were to run dominating set on the graph right now, we would be able to cover all the element vertices with any valid set cover, but we would have to pick every single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.

Proof of Correctness of Reduction

1. **If minimum set cover has a solution, then minimum dominating set that corresponds to it has a solution.**

Suppose we have some minimum set cover of size k. This will correspond to a dominating set of size k as well. For each set in our minimum set cover, pick the corresponding set vertex. It follows directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum set cover has a solution.**

Suppose we have some dominating set D of size k. We can find a set cover of size $\leq k$. To do this, we will construct a new dominating set D' that contains only set vertices. Include every set vertex in D in D'. For each vertex in our dominating set that is an element vertex, pick any random neighboring set vertex and add it to D'. Observe that $|D'| \leq |D|$. Thus if there is a dominating set in G of size $\leq k$, there must be a set cover of size $\leq k$

Since this problem is in NP and is NP-Hard, it must be NP-Complete.

5 Approximating Independent Set

In the maximum independent set (MIS) problem, we are given a graph $G = (V, E)$, and our goal is to find the largest set of vertices such that no two vertices in the set are connected by an edge. For this problem, we will assume the degree of all vertices in G is bounded by d (i.e. $\forall v \in V,\ \deg(v) \leq d$).

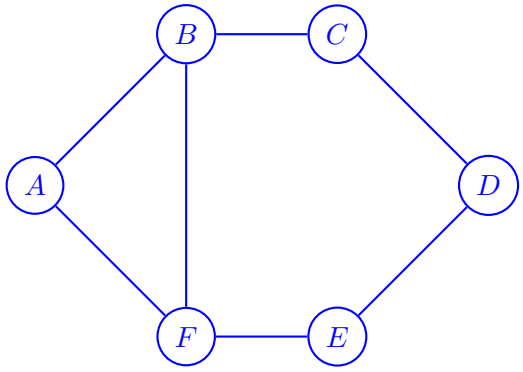
Consider the following greedy algorithm to approximate the maximum independent set:

```
1: procedure GREEDY-MIS(V, E)
2:     I ← ∅
3:     while G ≠ ∅ do
4:         choose a vertex v in G
5:         I = I ∪ {v}
6:         remove v and all its neighbors from G
7:     return I
```

(a) Provide an example where the greedy approximation does not give the optimal solution.

Solution:

Consider the graph below.



The maximum independent set would be of size 3, and can be composed of A, C, and E. However, if we follow the greedy algorithm, it's possible we choose vertex B, then one vertex from $\{D, E\}$, yielding an independent set of size 2.

(b) Provide an approximation ratio for the given greedy algorithm in terms of $|V|$, $|E|$, and/or d. Briefly justify your answer.

Solution: At each step of the while loop, one vertex is added to the independent set I. Also, at each step, at most $d + 1$ vertices are removed from G, since the maximum degree of a vertex in the graph is d, and both v and its neighbors are removed from G. Then, there are at least $|V|/(d + 1)$ iterations of the while loop. Since the maximum independent set OPT must be a subset of V, $|OPT| \leq |V|$. Then $|I| \geq \frac{|V|}{d+1} \geq \frac{|OPT|}{d+1}$. Hence, the approximation ratio is $d + 1$.