Autumn 1997–98, Prof. Boneh
Solution Set
CS 154, Problem Set 7

7.1 (3 points)

Problem: Dominating Set: **Given an undirected graph $G = (V, E)$ and an integer k, is there a subset S of V with $|S| < k$ such that for all $u \in V \setminus S$ there exists a $v \in S$ with $(u, v) \in E$? Prove that Dominating Set is $NP-$Complete.**

I'll be very complete (ha ha) in this solution, to try to make the process as clear as possible. We have to show 4 things. One, the problem is in NP. Two, we can convert an input to vertex cover to an input to the dominating set oracle in polynomial time. Three, that we can convert the solution to dominating set back to a solution to vertex cover in polynomial time. Four, that that the transformations preserve the size of the solution.

First we prove it is in NP. The certificate is the subset S that we claim dominates V; clearly it is polynomial in the size of the input. We verify S by going through each vertex in $V \setminus S$ and checking if there is an edge between the vertex and a vertex in S. Brute force, this takes time $|V||S||E|$ to verify every vertex is connected to some vertex in S, which is polynomial time. Since we can verify in polynomial time, we're in NP. We can also verify that $|S| < k$ quickly.

To prove completeness we reduce from the NP-complete problem vertex cover, which is a set S that touches every edge of G. Remember how this works: you give me a graph wanting a vertex cover and wanting it fast. I quickly convert the graph to some other graph G', pass it to the dominating set oracle, quickly get back a dominating set of G', and convert this into a vertex cover for G. We must preserve the quality of the solution: if G has a vertex cover of size k or less, G' must have a dominating set of size $f(k)$ or less.

What is the difference between vertex cover and dominating set? In dominating set, it's ok to have edges untouched by S as long as both their endpoints are. In vertex cover it's not. Our goal is to force dominating set to touch every edge in G. We do this by adding new vertices that can only be reached through an edge in G.

More specifically, we convert G by converting every edge AB into a triangle ABC. This requires adding, for each edge, a new vertex C and two new edges AC and BC. This takes time $2|E|$, which is polynomial, so the conversion is kosher. We hand this new graph to the dominating set oracle. It returns us a dominating set. I claim that at least one of A, B, or C must be in this dominating set. Why? By the definition of dominating set, if C is not in S itself, it must have a neighbor in S. But C's only neighbors are A and B.

We modify the dominating set a little, replacing every C with A (we could have chosen B). The result is still a dominating set, since A touches every vertex that C touches.

I claim that this dominating set is actually a vertex cover of the original graph. Note that S consists only of vertices in the original graph G, not added vertices. Furthermore, as the previous discussion showed, every edge AB has either A or B in S. Thus it's a vertex cover.

Finally, we show the transformation is size preserving: if G' has a dominating set of size $< k$, G has a vertex cover of size $< k$. This is easy to see because each vertex in the dominating set is also a vertex in the vertex cover. The only difference is if C is in the dominating set, we replace it by A. But if A is already in the dominating set, then the vertex cover we get will be smaller than the dominating set. But this is ok (intuitively, it would be bad if the oracle gave us a small solution that we blew up to a huge solution in the transformation. But here we're making an already-small-enough solution even smaller).

7.2 (3 points)

PROBLEM: INDEPENDENT SET: **Given an undirected graph $G = (V, E)$ and an integer $k > 2$, is there a subset S of V with $|S| > k$ such that for all $u, v \in S$ there is no edge between u and v? Prove that if $P \neq NP$ then Independent Set is not in P.**

The problem is phrased in this weird way just to be confusing. If we prove INDEPENDENT SET is NP-complete, then we're done: we've already shown that if an NP-complete problem can be solved in polynomial time, $P = NP$. We're just asked to prove the contrapositive of that.

First, INDEPENDENT SET in in NP: the certificate is the subset S, and we just verify in $|S|^2|E|$ time that no edge connects two vertices in S. We also check that $|S| > k$. (Why do we have this k, btw? Because then it's a decision problem. We don't know how to make TMs compute functions like "return the largest independent set in G.")

To prove it's NP-complete, we reduce from CLIQUE. Given a graph G you want to find the clique of, and a size limit k, pass the graph $\overline{G}$ and the limit k to the fast oracle for INDEPENDENT SET. The set S returned by the oracle is, I claim, a clique. (NB: $\overline{G}$ is the complement of G, that is, replace all edges with non-edges and vice-versa. This transformation takes at most $|V|^2|E|$ time and is thus polynomial.)

This is actually easy to see. Suppose S is an independent set in $\overline{G}$. This means that no two nodes in S are connected by an edge. But, by definition, that means *all* two nodes in S are connected by an edge in G, the complement of $\overline{G}$! That is, S is a clique in G. Clearly, the size of the independent set and the size of the clique are the same (so there's a clique of size $> k$ iff the oracle can find an independent set of size k), and the transformation of the solution takes polynomial (actually constant) time, so we've got a valid reduction, proving that INDEPENDENT SET is NP-complete.

7.3 (4 points)

PROBLEM: ***coNP*** **is the set of languages L for which $\bar{L} \in NP$. (a) Prove that $L \in coNP$ if and only if there exists a deterministic polynomial time verifier M such that $w \in L \iff \forall c \in \Sigma^* : M$ accepts (w, c) (b) Prove that $\overline{\text{CNF} - \text{SAT}} = \Sigma^* \setminus \text{CNF} - \text{SAT}$ is *coNP* complete. That is, all languages $L \in coNP$ satisfy $L \leq_P \overline{\text{CNF} - \text{SAT}}$.**

(a) First, the if direction. Assume L is in $coNP$. Then $\overline{L}$ has a poly-time verifier $\overline{M}$. That means that for every $w \notin L$ there is no certificate c such that M accepts w, c. Convert $\overline{M}$ to M by swapping the q_a and q_r states. If $w \notin \overline{L}$, $\overline{M}$ rejects all would-be certificates, which means M accepts them all. If $w \in \overline{L}$, $\overline{M}$ accepts at least one certificate, so M rejects at least one certificate. Thus M is the type of verifier required by the problem.

The other direction is the same. If we have a verifier M for L, swap q_a and q_r. By the abvoe argument, the new machine is an NP verifier for $\overline{L}$, which means by defintion that L is in $coNP$.

(b) This is just like Cook's theorem. Since L is in $coNP$, it has a verifier M as in part **(a)** that runs in polynomial time. We convert M into a verifier for $\overline{L}$ as in part **(a)**.

We use the same tableau as in Cook's theorem. Remember, there was a satisfying formula in Cook's theorem iff some witness (ie assignment) led to q_a (that is, led to a satisfied formula). But for this problem, a word is in L if there is *no* certificate accepted by the modified M. That is, there is no satisfying assigment for the tableau. But this means exactly the the formula that comes out of Cook's theorem is in $\overline{CNF - SAT}$. Since Cook's theorem takes polynomial time, and converting M takes polynomial time, the whole reduction takes polynomial time, and thus, there is a poly-time reduction from every problem in $coNP$ to $\overline{CNF - SAT}$.