1

(20 points) Show that the language $HALT$ is $NP$-hard. Is it $NP$-complete?

Note that a problem $H$ is $NP$-hard when every problem $L \in NP$ can be reduced in polynomial time to $H$. We also recall that any problem $L \in NP$ is reducible in polynomial time to $SAT$ the Boolean satisfiability problem.

Thus, we could reduce $SAT$ to $HALT$ in polynomial time, which suffices to show that $HALT$ is $NP$-hard.

Given any CNF formula $\phi$, we create an input pair $\langle M_\phi, \alpha \rangle$ to $HALT$ as follows. Let $\alpha = 0$, and the description of TM $M_\phi$ is

1. Try all possible assignments of truth values to the variables of $\phi$.

2. If any assignment makes $\phi$ TRUE, accept; else, loop forever.”

If $\phi$ is satisfiable, then $M_\phi$ will accept $\alpha$ and thus halts (since it tries all possible truth value assignments). Otherwise, $M_\phi$ will loop forever on $\alpha$. Thus $\phi \in SAT$ iff $\langle M_\phi, \alpha \rangle \in HALT$. Note that the reduction, which given $\phi$ writes the description $M_\phi$, runs in polynomial time. Therefore $SAT \leq_p HALT$ which shows that $HALT$ is $NP$-hard.

Note that $HALT$ is not $NP$-complete, since $HALT$ is undecidable, but $HALT \in NP$ would imply that $HALT$ could be decided in exponential time (since $NP \subseteq EXP$).

2

(20 points) Show that, if $P = NP$, then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$, is $NP$-complete. Here, $\Sigma$ is the alphabet, and you may assume that it is $\{0, 1\}$.

Suppose $P = NP$. Take any language $L_1 \in NP$, and any language $L_2 \neq \emptyset, L_2 \neq \Sigma^*$ in $P$, we show $L_1 \leq_p L_2$. Since $L_2 \neq \emptyset, L_2 \neq \Sigma^*$, there must be two strings $l_{in} \in L_2, l_{out} \notin L_2$ by giving a polynomial time function $f$. Since $L_1 \in NP \Rightarrow L_1 \in P$, there must be some polynomial time decider $D$ for $L_1$. Given any input $w$ to $L_1$, we map $w$ to $f(w)$ by first computing $D(w)$ and outputting $l_{in}$ upon acceptance, and $l_{out}$ otherwise. $D$ runs in
polynomial time, so this map takes polynomial time. We have that \( w \in L_1 \Rightarrow f(w) = l_{in} \in L_2 \) and \( w \notin L_1 \Rightarrow f(w) = l_{out} \notin L_2 \). Thus \( w \in L_1 \) if and only if \( f(w) \in L_2 \). Also, \( L_2 \in \text{NP} \), so \( L_2 \) is \text{NP} complete.

Note that we could not do the same for \( A = \emptyset \) or \( A = \Sigma^* \) since for the reduction to work we would need to map from \( w \in L_1 \) to \( f(w) \in L_2 \) and from \( w \notin L_1 \) to \( f(w) \notin L_2 \). This would be impossible if \( L_2 = \emptyset \) or \( L_2 = \Sigma^* \), since there is nothing in the set in the former case and nothing outside the set in the latter case.

3

(20 points) Let \( \phi \) be a 3\text{CNF}. An \( \neq \)-assignment to the variables of \( \phi \) is one where each clause contains two literals with unequal truth values.

(a) Show that any \( \neq \)-assignment automatically satisfies \( \phi \), and the negation of any \( \neq \)-assignment to \( \phi \) is also an \( \neq \)-assignment.

For any clause \( C \) of \( \phi \), the \( \neq \)-assignment will assign two literals in \( C \) with different truth values. This means that one of them is 1, meaning \( C \) is satisfied and thus \( \phi \) is satisfied. For the negation, every literal takes its negation. Thus, for each clause, the literals which had different truth values originally will be flipped, but they will still have different truth values, so the negation is still a \( \neq \)-assignment.

(b) Let \( \neq \text{-SAT} \) be the collection of 3\text{CNF}s that have an \( \neq \)-assignment. Show that we obtain a polynomial time reduction from 3\text{SAT} to \( \neq \text{-SAT} \) by replacing each clause

\[
c_i = (y_1 \lor y_2 \lor y_3)
\]

with the two clauses

\[
(y_1 \lor y_2 \lor z_i) \text{ and } (\overline{z_i} \lor y_3 \lor b),
\]

where \( z_i \) is a new variable for each clause \( c_i \) and \( b \) is a single additional new variable.

Let \( \phi \) be an input to the 3\text{SAT} problem. Use the replacements discussed to create \( \phi' \) from \( \phi \). It’s easy to see this transformation takes polynomial time since each clause only requires a constant number of operations. We now show that \( \phi \in 3\text{SAT} \) if and only if \( \phi' \in \neq \text{-SAT} \). Assuming that \( \phi \) is satisfied by some assignment, we know that any clause \( c_i = (y_1 \lor y_2 \lor y_3) \) must be 1 so \( y_1 \lor y_2 \lor y_3 = 1 \). To satisfy \((y_1 \lor y_2 \lor z_i) \) and \((\overline{z_i} \lor y_3 \lor b) \) we can set \( z_i = (y_1 \lor y_2) \) and \( b = 0 \) (note that \( b \) is set to 0 for all clauses). This is a \( \neq \)-assignment since \( z_i = (y_1 \lor y_2) \) and \( b = 0 = (\overline{z_i} \lor y_3) \), which by (a) automatically satisfies the two new clauses.

Now assume \( \phi' \) has a \( \neq \)-assignment. If \( b = 1 \) in this assignment, by (a) we take the negation of the assignment to get another \( \neq \)-assignment, which has \( b = 0 \). If \( b = 0 \) in the assignment, then we claim that the assignments to \( y_1, \ldots, y_n \) give a satisfying assignment for \( \phi \). To see this, note that for each clause \( c_i \), the literals \( y_1, y_2, y_3 \) cannot be all 0, since if this happens then no \( z_i \) can simultaneously satisfy \((y_1 \lor y_2 \lor z_i) \) and \((\overline{z_i} \lor y_3 \lor b) \). Thus \( c_i \) is satisfied and \( \phi \) is also satisfied. This shows that 3\text{SAT} reduces to \( \neq \text{SAT} \) in polynomial time.
(c) Conclude that \( \neq \text{SAT} \) is \( \text{NP} \)-complete.

Due to the reduction in part (b), we only need to show that \( \neq \text{SAT} \) is in \( \text{NP} \). For an input \( \phi \), we can make a certificate \( c \) to be an assignment to the variables. \( \phi \in \neq \text{SAT} \) iff there exists a \( \neq \) assignment, so we can just check \( c \) to see if it is a \( \neq \)-assignment by checking if each clause has two literals of different values. This can be done in polynomial time, so \( \neq \text{SAT} \) is in \( \text{NP} \), and further \( \neq \text{SAT} \) is \( \text{NP} \)-complete.

4

(20 points) Let DOUBLE-SAT = \{ \( \phi \) | \( \phi \) is a CNF that has at least two satisfying assignments\}. Show that DOUBLE-SAT is \( \text{NP} \)-complete.

We reduce 3SAT to DOUBLE-SAT.

Given an input 3CNF \( \phi \), we create a new clause \( C = u \lor \bar{u} \), where \( u \) is a newly added variable. Let \( \phi' = f(\phi) = \phi \land C \). So the function \( f(\cdot) \) can be computed in polynomial time. We claim that

\[ \phi \in 3\text{SAT} \iff \phi' \in \text{DOUBLE-SAT}. \]

If \( \phi \in 3\text{SAT} \), assume a satisfying assignment is \( x = s \) where \( x \) is the variable vector of \( \phi \). Then we have at least two satisfying assignments for \( \phi' \), because \( x = s, u = 1 \) and \( x = s, u = 0 \) are both satisfying assignments for \( \phi' \).

If \( \phi' \in \text{DOUBLE-SAT} \), then we know that \( \phi \) must also be satisfiable because \( \phi \land C \) is satisfiable. So \( \phi \in 3\text{SAT} \).

This shows that DOUBLE-SAT is \( \text{NP} \)-hard.

It is also in \( \text{NP} \). The certificate is two satisfying assignments. The verifier is a TM \( M \) running in polynomial time, which tests whether the input 3CNF \( \phi \) is satisfied by both the two assignments and the two assignments are different.

This shows DOUBLE-SAT is in \( \text{NP} \). So it is \( \text{NP} \)-complete.

5

(20 points) A subset of the nodes of a graph \( G \) is a dominating set if every other node of \( G \) is adjacent to some node in the subset. Let

\[ \text{DOMINATING-SET} = \{ (G,k) | G \text{ has a dominating set with } k \text{ nodes} \}. \]

Show that it is \( \text{NP} \)-complete by giving a reduction from VERTEX-COVER. You can assume that \( G \) has no vertex with degree 0.

We reduce VERTEX COVER to DOMINATING-SET.

For every input \( x \) to VERTEX COVER, we will prove that an input \( x' \) to DOMINATING-SET can be constructed in polynomial time such that \( x \in \text{VERTEX COVER} \iff x' \in \text{DOMINATING-SET} \).
Let $x = \langle G, k \rangle$ where $G = (V, E)$. we can construct $x'$ in the following way. First we construct another graph $G' = (V', E')$ using $G$. Without lost of generality, assume that $G$ does not have vertices which has degree 0. Let $V' = V \cup V_E$ where $V_E$ is a set of new vertices such that each vertex $v_e \in V_E$ if and only if $e \in E$. On the other hand, let $E' = E \cup \tilde{E}$. Here $\tilde{E}$ includes the edges $(v_e, u)$ and $(v_e, w)$ for every $v_e \in V_E$ where we denote $e = (u, w)$. Let $x' = \langle G', k \rangle$.

Next we will show that $x \in \text{VERTEX COVER} \iff x' \in \text{DOMINATING-SET}$.

If $G$ has a vertex cover $C$ of size $k$, then $G'$ has a dominating set of size $k$. To see this, we claim the dominating set of $G'$ is exactly $C$. Let's consider every vertex in $G'$. For every $v \in V$, as we have assumed that $v$ has degree at least 1, it is connected with an edge of $E$. As $C$ is a vertex cover, this edge is connected with a vertex of $C$. So either $v$ is in $C$ or $v$ is adjacent to a vertex in $C$. For every $v_e \in V_E$, as the edge $e \in E$ is connected with a vertex in $C$, according to our definition of $v_e$, we know that $v_e$ is also adjacent to that vertex. As a result, $C$ is a dominating set of $G'$.

On the other hand, if $G'$ has a dominating set which is $D$ of size $k$, we claim that a vertex cover $C$ for $G$ of size $k$ can be found. Next we describe an algorithm to find $C$.

First let $C$ be the empty set. Then for each vertex $v$ in $D$, if $v \in V$, put $v$ into $C$. If $v \in V_E$, assuming it corresponds to the edge $e = (u, w)$, put either $u$ or $w$ into $C$.

As a result, $|C| \leq k$. Consider every edge $e = (u, w) \in E$. We know $v_e$ is adjacent to a vertex $v'_e$ of $D$. This vertex can only be $v_e$, $u$ or $w$. In any case, one of $u$ and $w$ is in $C$ which covers the edge $e$. As a result, $C$ covers all the edges. This means it is a vertex cover of size at most $k$ for $G$.

So DOMINATING-SET is NP-hard.

Next we show that it is in NP. For $x = \langle G = (V, E), k \rangle$ the certificate is a set $D$ of vertices such that $D \subseteq V$. Thus $|D| \leq |V|$. The verifier TM $M$ checks whether $D$ is a dominating-set of size $k$ for $G$. This checking can be accomplished in polynomial time as we can check the vertices one by one to make sure that each vertex is either in $D$ or adjacent to a vertex in $D$. According to the definition of NP, DOMINATING-SET is in NP.

As a result, DOMINATING-SET is NP-complete.