Homework 7

Aleck Zhao

April 26, 2018

1. Let HALT be the Halting language. Show that HALT is NP-hard. Is it NP-complete?

   Proof. We will construct a polynomial time reduction from 3SAT, which is known to be NP-complete. Given an instance S of 3SAT, let T be a TM that iterates over all possible assignments to this instance, so that it only halts if a satisfying assignment is found, otherwise it loops forever. Then if $\langle T, \langle S \rangle \rangle$ is in the HALT, that must mean there exists a satisfying assignment to S, and if not, there does not exist a satisfying assignment. Thus, this is a reduction from an instance of 3SAT to an instance of HALT, which is clearly polynomial time because converting T and S into representations can only take polynomial time. Thus, HALT is NP-hard.

   HALT is not NP-complete because it is not in NP. We know this because HALT is an undecidable language, and therefore no verifier can run in polynomial time. $\square$

2. Call graphs G and H isomorphic if the nodes of G can be reordered so that the graph G is identical to H. Let ISO$=\{\langle G,H\rangle : G,H$ are isomorphic$\}$. Show that ISO$\in$ **NP**.

Proof. Suppose we are given G and H and a certificate $c=\{i_1,\cdots,i_m\}$ of indices. Then we construct the verifier as $V(\langle G,H\rangle, c)$ as

(1) Suppose G has n vertices. First check if H also has n vertices. If not, reject.

(2) Now check if $\{i_1,\cdots,i_m\}$ is a permutation of $\{1,\cdots,n\}$. If not, reject.

(3) Now for each vertex v_j in H, take the map $v_i \mapsto v_{i_j}$. Now check if G and the transformed H are identical. If they are, accept, otherwise, reject.

Step (1) can be completed using a DFS, which takes $O(|V|+|E|)$ time. Step (2) can be completed using a sorting algorithm, which takes $O(n^2)=O\left(|V|^2\right)$ time. Step (3) can be completed by just checking every edge and every vertex, which takes $O(|V|+|E|)$. Thus, this verifier runs in polynomial time in the size of the inputs. It is clearly a correct verifier since it checks everything that needs to be checked, so ISO is in NP. $\square$

3. Show that, if $\mathbf{P} = \mathbf{NP}$, then every language $A \in \mathbf{P}$, except $A = \varnothing$ and $A = \Sigma^*$, is NP-complete.

   Proof. If $\mathbf{P} = \mathbf{NP}$, then if $A \in \mathbf{P}$ we have $A \in \mathbf{NP}$. Now, to show that A is NP-hard, we need to show that any $B \in \mathbf{NP}$ can be solved in polynomial time using an oracle for A. Since $\mathbf{P} = \mathbf{NP}$, this means $B \in \mathbf{P}$ so every language can be solved in polynomial time given an oracle for A (that we wouldn't even need to use). Thus, A is NP-hard, and thus A is NP-complete. $\square$

4. Let ϕ be a 3CNF. An $\neq$-assignment to the variables of ϕ is one where each clause contains two literals with unequal truth values.

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

Proof. If $(x \vee y \vee z)$ is a clause in a $\neq$-assignment, where WLOG x and y have unequal truth values, this clause evaluates to 1. Since all clauses satisfy this property, combining all clauses will also yield a truth value of 1, and thus satisfy ϕ.

If we negate the $\neq$-assignment, consider the clause $(x \vee y \vee z)$ in the original, which becomes $(\neg x \vee \neg y \vee \neg z)$. If WLOG x and y had unequal truth values in the original, then $\neg x$ and $\neg y$ have unequal truth values, so each clause still satisfies the property of being a $\neq$-assignment. □

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

$$c_i = (y_1 \vee y_2 \vee y_3)$$

with the two clauses

$$(y_1 \vee y_2 \vee z_i) \text{ and } (\bar{z}_i \vee y_3 \vee b)$$

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

Proof. $(\implies)$: Consider a satisfying assignment to clause i being $(y_1 \vee y_2 \vee y_3)$. Take $b = 0$. Then if y_1, y_2 are both 0, we must have y_3 be 1 in order for the clause to be satisfied, so we can take $z_i = 1$ and construct the two clauses $(y_1 \vee y_2 \vee 1)$ and $(0 \vee y_3 \vee 0)$ which are both valid and satisfying $\neq$-assignments.

Otherwise, one of y_1, y_2 is not 0, so we can take $z_i = 0$, so we can construct the two clauses $(y_1 \vee y_2 \vee 0)$ and $(1 \vee y_3 \vee 0)$, which are both valid $\neq$-assignments. This is clearly polynomial time since we have only doubled the number of clauses, so if there exists a satisfying assignment to the original 3SAT, there exists a satisfying $\neq$-assignment.

$(\impliedby)$: Consider a satisfying $\neq$-assignment to clauses i being $(y_1 \vee y_2 \vee z_i)$ and $(\bar{z}_i \vee y_3 \vee b)$. If one of y_1, y_2, or y_3 is not 0, then the clause $(y_1 \vee y_2 \vee y_3)$ would be satisfied. Otherwise, if they are all 0, then by part (a), negating this $\neq$-assignment will still be satisfying, which means one of $\bar{y}_1, \bar{y}_2$, or $\bar{y}_3$ would not be 0, and thus $(\bar{y}_1 \vee \bar{y}_2 \vee \bar{y}_3)$ is a satisfying assignment for 3SAT. Clearly this is polynomial time, so if there exists a satisfying assignment to the $\neq$SAT, there exists a satisfying assignment for SAT. □

(c) Conclude that $\neq$SAT is NP-complete.

Proof. Clearly, if given an assignment, we can determine if it is a valid $\neq$-assignment in polynomial time (just go through each clause and check), and we can also determine if it is satisfying by simply evaluating, so $\neq$SAT is in NP.

Since 3SAT is NP-complete and there exists a polynomial time reduction from 3SAT to $\neq$SAT, it follows that $\neq$SAT is NP-hard, and thus NP-complete. □