Homework 8 Solutions

Prepared by Khanh Do Ba

Prof. Amit Chakrabarti
Computer Science Department
Dartmouth College

1. **(Proof that P $\neq$ NP?)**

   The error lies in the definition of the "time complexity" of SAT. The time complexity of SAT is the running time of the *best* decider for SAT. The given algorithm is just one, and might not be the best one. Therefore, all it tells us is that the time complexity of SAT is *at most* exponential.

2. **(DoubleSAT is NP-complete)**

   Proof. To see that DoubleSAT is in NP, we use the standard guess-and-check approach: guess two distinct assignments to ϕ and accept if they both satisfy ϕ.

   To prove that DoubleSAT is NP-hard, we reduce SAT to it, as follows.

```
"On input $\phi$:
  1 Let $a$ be a new variable not in $\phi$.
  2 Let $\phi' = (\phi \wedge a) \vee (\phi \wedge \overline{a})$.
  3 Output $\phi'$."
```

   Suppose A is an assignment that makes ϕ true. Then extending A by either $a = T$ or $a = F$ makes ϕ' true as well, so that ϕ' has at least two satisfying assignments. On the other hand, if ϕ cannot be satisfied, then regardless of what we assign to a, neither $(\phi \wedge a)$ nor $(\phi \wedge \overline{a})$ can be satisfied, whence ϕ' is unsatisfiable as well. Therefore this is a correct reduction, and we are done. □

3. **(Knapsack is NP-complete)**

   3.1. Consider the following counterexample: $\langle 3, 3, 2, 2, 5, 3, 3, 4, 6 \rangle$. That is, we have three objects: A, B, and C, where A has weight 3 and value 5, while B and C both have weight 2 and value 3; the weight limit of our knapsack is 4; and our target loot is 6. Now, since $5/3 > 3/2$, the greedy burglar (algorithm) will pick A first. But after that, he will have only 1 weight unit left in his knapsack, so will not be able to take anything else, giving him a total loot of only $5 < 6$. On the other hand, the smart burglar would have taken B and C, giving him a total loot of 6.

   3.2. *Proof.* To nonderterministically decide Knapsack, simply guess a subset of items and check if it satisfies the weight and value requirements. This can certainly be done in polynomial time, so Knapsack is in NP.

   To prove that it is NP-hard, we reduce Subset-Sum to it. Recall that Subset-Sum consists of pairs $\langle S, t \rangle$, where S is a multiset of positive integers, some subcollection of which adds up to t. Our reduction is as follows.

```
"On input $\langle S, t \rangle$:
  1 Let $S = \{x_1, \ldots, x_n\}$.
  2 Output $\langle n, x_1, \ldots, x_n, x_1, \ldots, x_n, t, t \rangle$."
```

   Observe that

$$\begin{aligned}
\langle S, t \rangle \in \textsc{Subset-Sum} &\iff \exists T \subset \{1, \ldots, n\} \text{ such that } \sum_{i \in T} x_i = t \\
&\iff \exists T \subset \{1, \ldots, n\} \text{ such that } \left(\sum_{i \in T} x_i \leq t \bigwedge \sum_{i \in T} x_i \geq t \right) \\
&\iff \langle n, x_1, \ldots, x_n, x_1, \ldots, x_n, t, t \rangle \in \textsc{Knapsack}.
\end{aligned}$$

Hence this is a correct reduction, and we are done. □

4. **(MaxCut is NP-complete)**

4.1. (a) *Proof.* Let A be a $\neq$-assignment to ϕ. Then A assigns either one or two true literals to each clause of ϕ. Consider $\overline{A}$, the negation of A. Every clause where A assigned one true, $\overline{A}$ must now assign two trues, and every remaining clause, $\overline{A}$ must now assign one true. It follows that $\overline{A}$ assigns either one or two true literals to every clause, and thus is a $\neq$-assignment as well. □

(b) *Proof.* Suppose we take a 3cnf-formula ϕ and replace each clause $c_i = (y_1 \vee y_2 \vee y_3)$ with the two clauses $(y_1 \vee y_2 \vee z_i)$ and $(\overline{z_i} \vee y_3 \vee b)$ to produce ϕ'.

To see that this is a reduction from 3SAT to $\neq$SAT, suppose ϕ had a satisfying assignment A. Then for each clause $c_i = (y_1 \vee y_2 \vee y_3)$, at least one of y_1, y_2, y_3 must have been true under A. To get a $\neq$-assignment for ϕ', first extend A with $b = F$. Now, if either y_1 or y_2 is true, extend A with $z_i = F$. Otherwise, if y_3 is true, extend A with $z_i = T$. In either case, we make both clauses true, but with at least one false literal in each. Since we can do this for every i, we have a $\neq$-assignment for ϕ'.

Conversely, suppose ϕ has no satisfying assignment, and let A' be a satisfying $\neq$-assignment to ϕ'. We can restrict A' to the variables of ϕ to get an assignment A. But A must assign all three literals false in some clause c_i of ϕ, which forces A' to have assigned $z_i = T$ and $b = T$. Now, suppose A assigns all three literals true for some clause c_j. Then in the corresponding two clauses in ϕ', no matter what value z_j takes, at least one must have three true literals, contradicting the assumption that A' is an $\neq$-assignment. Hence such an assignment cannot exist, and we are done. □

(c) *Proof.* Since we know that 3SAT is NP-hard, the above reduction tells us that $\neq$SAT is NP-hard as well. To see that it is in NP, observe that we can decide it by simply guessing an assignment, and checking that it is indeed $\neq$. It follows that $\neq$SAT is NP-complete. □

4.2. *Proof.* We can decide Max-Cut by simply guessing a cut and counting its size. This is clearly polynomial, so Max-Cut is in NP. We show that it is NP-hard by a reduction from $\neq$SAT, as follows.

```
"On input $\phi$:
  1 Convert each clause $(x \vee y \vee z)$ into a triangle with vertices $x, y, z$ and edges
    $\{x, y\}, \{x, z\}, \{y, z\}$.
  2 For every pair of vertices $x$ and $\overline{x}$, add edge $\{x, \overline{x}\}$.
  3 Let $k$ be the number of edges added in Line 2, and $G$ be the resulting graph.
  4 Output $\langle G, k + 2c\rangle$, where $c$ is the number of clauses in $\phi$."
```

Note that this construction is identical to the one you saw in class used to reduce 3SAT to IS, with the addition only of the $k + 2c$. Now, suppose A is an $\neq$-assignment to ϕ. Consider the cut of G consisting of $T = \{\text{vertices assigned true}\}$ and $F = \{\text{vertices assigned false}\}$. Observe that all k "conflict" edges must cross the cut for A to be an assignment at all. Moreover, since A is an $\neq$-assignment, two edges from each of the c triangles must cross the cut. This gives us a total of $k + 2c$ edges, as desired.

Conversely, suppose (T, F) is a cut of G of size $\geq k + 2c$. Note that at most two edges in any triangle can cross the cut, so those can at best give us a contribution of $2c$ edges. Thus, the remaining k "conflict" edges must all cross, giving us a total, in fact, of exactly $k + 2c$.

But this gives us an $\neq$-assignment to ϕ, with all vertices in T assigned true and those in F assigned false, so we are done. $\square$

5. **(RSA cryptography and P = NP)**

First, we can restate the problem of breaking the RSA cryptosystem as that of factoring a product of two primes; that is, given an input positive integer $n = pq$, p, q prime, we must find and output either p or q.

Consider the following decision problem: given positive integers n, k, where $k < n$, does n have a factor p such that $2 \leq p \leq k$? This can be solved in nondeterministic polynomial time by guessing such a p and dividing n by it to see if it is a factor, hence it is in NP. But if P = NP, then it is in P as well, so we can use a black-box polytime decider for it to do a binary search on $\{1, \ldots, n\}$ and find a factor in $O(\log n)$ iterations.

Note that a brute force search of $\{1, \ldots, n\}$ will *not* work since our running time needs to be polynomial in the *size of the input*, that is, $\log n$.