Solutions to Homework 8

We may grade a **subset of the assigned questions**, to be determined after the deadline, so that we can provide better feedback on the graded questions.
Unless otherwise stated, each question requires sufficient justification to convince the reader of the correctness of your answer.
For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must write your own solution for all problems and may not look at any other student's write-up.

0. If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. (a) Prove the transitive property for polynomial-time mapping reductions:

$$A \le_p B \text{ and } B \le_p C \implies A \le_p C.$$

(b) Prove that if $A \le_p B$ and $B \in \mathsf{NP}$, then $A \in \mathsf{NP}$.

(c) Prove that if $A \le_p B$, then $A \le_T B$.

Solution:

(a) Suppose that $A \le_p B$ and $B \le_p C$. By definition, this means there exist efficiently computable functions f and g such that

- $x \in A \iff f(x) \in B$ and
- $y \in B \iff g(y) \in C$.

We show that $A \le_p C$ by exhibiting an efficiently computable function h such that $x \in A \iff h(x) \in C$. The function h is simply the composition $g \circ f$, i.e., $h(x) = g(f(x))$. By the above properties of f and g, we have

$$x \in A \iff f(x) \in B \iff g(f(x)) \in C,$$

so it immediately follows that $x \in A \iff h(x) \in C$, as required.

Lastly, we confirm that $h = g \circ f$ is efficiently computable. By assumption, $f(x)$ is computable in some $O(|x|^c)$ time, and $g(y)$ is computable in some $O(|y|^d)$ time, where c, d are some constants. If $y = f(x)$, then its length $|y| = O(|x|^c)$, because f cannot output a string that is longer than its running time. So, $g(y)$ is computable in time $O(|y|^d) = O((|x|^c)^d) = O(|x|^{cd})$. Thus, $h(x) = (g \circ f)(x)$ is computable in time $O(|x|^c + |x|^{cd})$, which is polynomial in $|x|$ because c and cd are constants.

(b) Suppose that $A \le_p B$ and $B \in \mathsf{NP}$. Let f be an efficiently computable function for which $x \in A \iff f(x) \in B$ (as guaranteed by the first hypothesis), and let V_B be an efficient verifier for B (as guaranteed by the second hypothesis). Then we have $y \in B \iff$ there exists some c such that $V_B(y, c)$ accepts.

We construct an efficient verifier V_A for A, which works as follows: on input (x, c), run $V_B(f(x), c)$ and output the same decision.

[Notice that this V_A does something that may initially seem strange: to verify that a string is in A, it expects a "B-type certificate"! (For example, if A = 3SAT and B = VERTEX-COVER, the certificate for a formula ϕ would be a vertex cover of an associated graph.) But a closer look reveals that this makes sense: to verify that $x \in A$, V_A will actually verify that $f(x) \in B$ using V_B, so it needs a certificate for the latter to do so. The following analysis shows that this V_A satisfies the formal requirements of a verifier for language A.]

To show that this V_A is a valid verifier for A, we need to show that (1) it is efficient, and (2) for all x, we have $x \in A \iff$ there exists some c such that $V_A(x, c)$ accepts.

For (1), V_A is clearly polynomial time with respect to its first argument because f is computable in polynomial time, and V_B is polynomial time in its first argument. (This is essentially the same argument as in the previous part.)

For (2), by the above hypotheses and the definition of V_A, we have

$$x \in A \iff f(x) \in B \iff \exists\, c : V_B(f(x), c) \text{ accepts} \iff \exists\, c : V_A(x, c) \text{ accepts.}$$

So, $x \in A \iff$ there exists a certificate c such that $V_A(x, c)$ accepts, as required.

(c) Suppose $A \leq_p B$, and let f be an efficiently computable function such that $x \in A \iff f(x) \in B$. We prove $A \leq_T B$ via the following Turing reduction from A to B. Let D_B be some black-box decider for B, and construct a decider D_A for A (that uses D_B) as follows: on input x, run $D_B(f(x))$ and output the same decision. The step of computing $f(x)$ is doable by a Turing machine because f is an (efficiently) computable function. Clearly, D_A is a decider for A because it halts on every input, and $x \in A \iff f(x) \in B \iff D_B(f(x))$ accepts $\iff D_A(x)$ accepts.

2. Recall the 0-1 Knapsack Problem: we are given two length-n arrays containing positive integer weights $W = (w_1, w_2, \dots, w_n)$ and values $V = (v_1, v_2, \dots, v_n)$, and a weight capacity $C \in \mathbb{N}$. The goal is to select items having maximum total value whose total weight is below the threshold. We can define this as a decision problem by introducing a threshold K and asking whether a value of at least K can be achieved (subject to the weight constraint):

$$\textsc{Knapsack} = \{(W, V, C, K) : \exists\, S \subseteq \{1, \dots, n\} \text{ such that } \sum_{i \in S} W[i] \leq C \text{ and } \sum_{i \in S} V[i] \geq K\}.$$

In this problem you will show that Knapsack is NP-Complete.

(a) Prove that Knapsack $\in$ NP.

(b) Prove that Knapsack is NP-Hard, by showing that Subset-Sum $\leq_p$ Knapsack. Conclude that Knapsack is NP-Complete.

(c) Recall that we previously gave a dynamic programming algorithm that solves Knapsack in $O(nC)$ time. Does this prove that P = NP? Why or why not?

Solution:

(a) In order to show that KNAPSACK $\in$ NP, we must construct a valid verifier. For a KNAPSACK instance, the verifier expects the certificate to be a legal subset S of chosen items that have total value at least K. Specifically, our verifier works as follows:

```
1: function VERIFIER($(W, V, C, K), S$)
2:     if $S \not\subseteq \{1, \ldots, n\}$ then
3:         reject
4:     if $\sum_{i \in S} W[i] \leq C$ and $\sum_{i \in S} V[i] \geq K$ then
5:         accept
6:     reject
```

- If $(W, V, C, K) \in$ KNAPSACK, then by definition, there exists some $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} W[i] \leq C$ and $\sum_{i \in S} V[i] \geq K$. Therefore, running our verifier on this instance and this S as the certificate will cause it to accept, as desired.
- If $(W, V, C, K) \notin$ KNAPSACK, then by definition, there does not exist any subset S meeting the above requirements. Because the verifier accepts only if the presented certificate is such a subset S, the verifier will reject this (W, V, C, K) for *all* certificates, as required.

Finally, it is apparent that the verifier is efficient with respect to its first input, as required.

(b) We show that SUBSET-SUM $\leq_p$ KNAPSACK, i.e., we exhibit an efficiently computable function f such that

$$(A, s) \in \text{SUBSET-SUM} \iff f(A, s) \in \text{KNAPSACK}.$$

Define $f(A, s) = (W = A, V = A, C = s, K = s)$. In words: given a subset-sum instance with integers A and target sum s, we map it to a knapsack instance with an item for each number in A, whose weight and value both equal that number; moreover, the knapsack capacity C and target value K both equal the target sum s.

We show that f has the required properties. It is clearly efficiently computable. For the $\Rightarrow$ direction, if some subset of A sums to s, then the corresponding collection of knapsack items has total weight $s \leq C$ (the knapsack capacity), and has total value $s \geq K$ (the target value); in fact, both of these are equalities. For the $\Leftarrow$ direction, if some subset of knapsack items has total weight $\leq C = s$ and total value $\geq K = s$, then because each item's weight equals its value, the total weight and value are both exactly s. This means that the corresponding subset of numbers in A sums to s, as required.

(c) Despite it running in time $O(nC)$, the DP algorithm is actually *not* efficient. Recall that C is a number, and is represented in binary using some t digits, e.g., $t = n$ or more. So the value of C can be as large as $2^t - 1$, meaning that the DP algorithm's running time is *exponential* in the input length. So, this does not show that $\mathsf{P} = \mathsf{NP}$. (Nor does it show that $\mathsf{P} \neq \mathsf{NP}$. We have only shown that a *particular* algorithm for Subset-Sum takes exponential time; we have not ruled out the possibility of a polynomial-time algorithm.)

3. Recall that Clique $\in \mathsf{NP}$, where Clique is defined as:

$$\text{Clique} = \{\langle G, k\rangle : G = (V, E) \text{ has a clique of size } \geq k\}.$$

(Recall: A *clique* of an undirected graph $G = (V, E)$ is a subset $K \subseteq V$ such that every pair of vertices in K has an edge between them.) Consider the following variant of Clique:

Half-Clique $= \{\langle G\rangle : |V| = 2n$, and G has a clique of size $\geq n\}$.

Prove that Half-Clique is NP-complete.

Solution:

Half-Clique is NP-complete. First, observe that Half-Clique is in NP because we can construct a verifier that checks to see that its certificate represents a valid clique containing at least half of the vertices in its input graph. This is efficient because if the input graph has $2n$ vertices, then checking every pair of vertices in the certificate will take $O(n^2)$ time (assuming the input is given as an adjacency matrix), which is polynomial.

Now we show Clique $\leq_p$ Half-Clique. To do this, we will construct a mapping $f(\langle G, k\rangle)$ that, given an instance $\langle G, k\rangle$ of Clique where $G = (V, E)$, outputs an instance $\langle G'\rangle$ of Half-Clique such that $\langle G, k\rangle \in$ Clique $\iff \langle G'\rangle \in$ Half-Clique. Below, we show two possible valid solutions.

Solution #1: Define $f(\langle G, k\rangle) = \langle G'\rangle$ where G' is constructed by adding $|V| - k$ vertices that are all adjacent to each other and to all other vertices in G, and also adding k isolated vertices that have no incident edges.

Analysis: f can be computed in polynomial time because it adds a total of $|V|$ vertices and $O((|V| - k)^2 + |V|(|V| - k)) = O(|V|^2)$ edges to G, which can be done in polynomial time.

Now we show that $\langle G, k\rangle \in$ Clique $\iff \langle G'\rangle \in$ Half-Clique. Observe that if G has $n = |V|$ vertices, then G' will have $2n$ vertices, so this is equivalent to showing that G has a clique of size k if and only if G' has a clique of size n.

($\Rightarrow$) Suppose G has a clique of size k. Then these k vertices from G, together with the $n - k$ vertices that the mapping added to G' that are adjacent to all vertices in G, forms a clique of size n in G'.

($\Leftarrow$) Suppose that G' has a clique of size n. Now, at most $n - k$ of the vertices in this clique can be vertices that were added to G by the mapping to form G', which means that at least k of the vertices in the clique must be vertices that originally came from G. But this means that G has a clique of size at least k.

Solution #2: Define $f(\langle G, k\rangle) = \langle G'\rangle$ where G' is constructed depending on the following cases.

Case 1: If $k < |V|/2$, then G' is constructed by adding $|V| - 2k$ vertices that are all adjacent to each other and to all other vertices in G.

Case 2: If $k > |V|/2$, then G' is constructed by adding $2k - |V|$ isolated vertices to G.

Case 3: If $k = |V|/2$, then $G' = G$.

The idea here is that we are "padding" the graph G by adding additional vertices so that a k-clique in G corresponds to a clique of half the number of vertices in G'.

Analysis: f can be computed in polynomial time because it adds at most $||V| - 2k|$ vertices and $O(|V|^2)$ edges to G. We show that $\langle G, k\rangle \in \textsc{Clique} \iff \langle G'\rangle \in \textsc{Half-Clique}$. To do this, we will consider the three cases separately.

Case 1: If $k < |V|/2$, then G' will have $2|V| - 2k$ vertices. Thus we need to show that G has a k-clique if and only if G' has a $(|V| - k)$-clique.

($\Rightarrow$) Suppose G has a k-clique. Then the same k-clique in G', along with the $|V| - 2k$ vertices that were added by the mapping, forms a clique of size $|V| - k$ in G'.

($\Leftarrow$) Suppose G' has a clique of size $|V| - k$. Then at most $|V| - 2k$ of those vertices could be the vertices added by the mapping, which means that at least k of those vertices must come from G. But this means that G has a clique of size at least k.

Case 2: If $k > |V|/2$, then G' will have $2k$ vertices. Notice that since all of the vertices added to G to form G' are isolated, then any k-clique in G corresponds to a k-clique in G' and vice-versa, hence $\langle G, k\rangle \in \textsc{Clique} \iff \langle G'\rangle \in \textsc{Half-Clique}$ as desired.

Case 3: If $k = |V|/2$, then $|V| = 2k$. Since we care about cliques of size k, the problem is already an instance of $\textsc{Half-Clique}$. Thus $\langle G, k\rangle \in \textsc{Clique} \iff \langle G\rangle \in \textsc{Half-Clique}$ as desired.

4. Suppose there are n students at Michigan and k clubs. Every student may join any number of clubs, possibly zero. Let $S = (S_1, \ldots, S_k)$ be a list of the students in each club, where each club i has a subset $S_i \subset [n]$ of students.

   Now given a tuple $q = (q_1, \ldots, q_k)$ of natural numbers, we want to enroll a subset of Michigan students in EECS 376 so that there are exactly q_i EECS 376 students in the ith club, for every $i \in [k]$.

   We say q is a *possible configuration* if this is possible. Note that not all qs are possible configurations. For example, if $S_i = [n]$ for every $i \in [k]$, i.e. every student joins every club, then the only possible q's are $(w, w, \ldots, w)$ for some $w \in \{0, 1, \ldots, n\}$ where w is the number of EECS 376 students.

   Concretely, define

   $$\begin{aligned}\textsc{Poss-Config} = \{\langle n, k, S, q\rangle : \ & q \text{ is a possible configuration, i.e. there exists } E \subset [n], \\ & \text{representing the set of EECS 376 students, such that} \\ & \text{for every } i \in [k],\ |S_i \cap E| = q_i\}.\end{aligned}$$

   Prove that 3SAT $\leq_p$ Poss-Config.

   Hints:

   (a) For each variable v_i in φ, allocate two students to represent when $v_i = T$ and $v_i = F$, respectively. How can you enforce that exactly one of them is enrolled in EECS 376 (perhaps by definition of a corresponding club)?

   (b) Assuming you've dealt with hint (a), how do you capture the constraint that all clauses evaluate to true (again by definition of a corresponding club)?

Solution: Let φ be a 3-CNF with n variables $(x_1, \ldots, x_n)$ and m clauses $(c_1, \ldots, c_m)$. Each clause $c_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$, where a literal is either a variable x_i or its negation $\overline{x_i}$. We construct a club configuration as follows.

- There are in total $2n + 2m$ students, denoted as $\{s_{x_1}, \ldots, s_{x_n}\} \cup \{s_{\overline{x_1}}, \ldots, s_{\overline{x_n}}\} \cup \{s_{c_1}, \ldots, s_{c_m}\} \cup \{s'_{c_1}, \ldots, s'_{c_m}\}$.
- There are in total $n + m$ clubs, denoted as $\{C_{x_1}, \ldots, C_{x_n}\} \cup \{C_{c_1}, \ldots, C_{c_m}\}$.
- For $i \in [n]$, club C_{x_i} has students $\{s_{x_i}, s_{\overline{x_i}}\}$.
- For $j \in [m]$, club C_{c_j} has students $\{s_{l_{j,1}}, s_{l_{j,2}}, s_{l_{j,3}}, s_{c_j}, s'_{c_j}\}$, where $c_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$. For example if $c_1 = x_1 \vee \overline{x_2} \vee \overline{x_6}$, then $C_{c_1} = \{s_{x_1}, s_{\overline{x_2}}, s_{\overline{x_6}}, s_{c_1}, s'_{c_1}\}$.
- The number of EECS 376 students in the number in the club C_{x_i} is $q_{x_i} = 1$ and the club C_{c_j} is $q_{c_i} = 3$ for any $i \in [n]$ and $j \in [m]$.

This mapping is clearly efficient. Thus it suffices to prove that

$$\varphi \text{ is satisfiable} \quad \iff \quad \langle 2n + 2m, n + m, C, q\rangle \text{ is a possible configuration.}$$

Assume φ is satisfiable and we fix a specific satisfying assignment of x. For each $i \in [n]$, if x_i is true, then s_{x_i} is an EECS 376 student and $s_{\overline{x_i}}$ is not; if x_i is false, then s_{x_i} is not an EECS 376 student and $s_{\overline{x_i}}$ is. Note that this guarantees the EECS 376 student count in the club C_{x_i} is 1. Furthermore, since every clause is satisfied, at least one literal in each clause is true. Thus C_{c_j} contains at least one s_l for some literal l (and at most three). The number of EECS 376 students can be made exactly 3 in every C_{c_j} by making s_{c_j} and s'_{c_j} EECS 376 students when necessary. Thus the configuration is a possible configuration.

On the other hand, if $\langle 2n+2m, n+m, C, q\rangle$ is a possible configuration, by design exactly one of s_{x_i} and $s_{\overline{x_i}}$ will be an EECS 376 student. We assign x_i to be true if s_{x_i} is in EECS 376 and $\overline{x_i}$ to be true otherwise. This assignment satisfies φ since for every clause j at least one student s_l is in the club C_{c_j}, which means l is true in c_j. Thus φ is satisfiable.

5. Let EXP be the class of all languages which are decidable in exponential time, i.e., in time $O(2^{n^k})$ for some constant k (where n is the length of input). Formally,

$$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}\left(2^{n^k}\right).$$

It remains unknown whether $\mathsf{NP} = \mathsf{EXP}$, but it is known that $\mathsf{P} \neq \mathsf{EXP}$. Prove that $\mathsf{NP} \subseteq \mathsf{EXP}$.

That is, for any language $L \in \mathsf{NP}$ and any input $x \in L$, provide a decider that runs in time $O(2^{n^k})$, where $n = |x|$ and k is some constant. Recall that any language $L \in \mathsf{NP}$ has a verifier $V(x, c)$ which runs in time $O(|x|^d)$ for a constant d. (As always, you should analyze the correctness and runtime of your decider.)

Solution: Consider an arbitrary language $L \in \mathsf{NP}$; we will show that $L \in \mathsf{EXP}$ (and thus $\mathsf{NP} \subseteq \mathsf{EXP}$).

By hypothesis, L has a verifier $V(\cdot,\cdot)$ which takes polynomial time in the length of the first input, and where $x \in L$ if and only if there exists a certificate c such that $V(x, c)$ accepts. A key observation is that because V runs in at most some polynomial $|x|^k$ time (for some constant $k \geq 1$), it can *read at most* $|x|^k$ *bits* of any given certificate c before halting. Thus, the accept/reject behavior of $V(x, c)$ is determined by x and *the first* $|x|^k$ *bits of* c.

With the above observation, to decide L we can simply do a brute-force search, running the verifier with every possible certificate of appropriate length, and accepting if any of these runs accepts. More precisely, we define the following program M:

```
1: function M(x)
2:     for each possible certificate c of length at most |x|^k do
3:         if V(x, c) accepts then
4:             accept
5:     reject
```

We claim that M decides L. If $x \in L$, then by definition there exists some certificate c

of length $\leq |x|^k$ such that $V(x, c)$ accepts. Thus, $M(x)$ will find such a c and accept. If $x \notin L$, then $V(x, c)$ rejects for every certificate c, so $M(x)$ rejects, as required.

We now analyze the running time of $M(x)$. There are fewer than 2^{t+1} bitstrings of length at most t, so the loop will iterate fewer than $2^{|x|^k+1}$ times. Each iteration runs the verifier, which takes at most $|x|^k$ time. Thus $M(x)$ takes time at most

$$2^{|x|^k+1} \cdot |x|^k = 2^{|x|^k+1+k\log|x|} = O(2^{|x|^{k+1}}).$$

Thus, M is an exponential-time decider for L, so $L \in \mathsf{EXP}$, as claimed.