Solutions to Discussion Worksheet 8

1 NP-Hardness and NP-Completeness

1. All/Some/No Questions

   (a) For (all/some/no) language(s) L in NP, L is decidable.

   > **Solution: All.**
   > Recall that $\mathsf{NP} \subseteq \mathsf{EXP}$, which means every language in the class NP is decidable in exponential time. Since by our assumption the language is in NP it has an efficient verifier. If we have an input of length n there are fewer than 2^{n+1} different bitstrings of that length. Therefore, as a verifier can only read a polynomial number of bits of the certificate ($|x|^k$) to remain efficient there are fewer than $2^{|x|^k+1}$ different certificates for our verifier. So we can decide the language by running our verifier on every possible certificate yielding a runtime of $O(|x|^k \cdot O(2^{|x|^k+1})) = O(2^{|x|^k+1+k\log_2(|x|)}) = O(2^{|x|^k})$.

   (b) For (all/some/no) NP-Hard languages L, L is decidable.

   > **Solution: Some.**
   > Languages such as SATand CLIQUEwhich are NP-Complete are NP-Hard as well as in NP. These languages are decidable in $O(2^{n^k})$ time because they are in NP. Hence NP-Hard languages can be decidable.
   > However, this is not always the case. In question 4 of this problem set, you will show that L_{EQ}, an undecidable language, is NP-Hard.

2. Prove the following claims:

   (a) The languages $\emptyset$ and Σ^* are not NP-hard.

   > **Solution:** We argue that $\emptyset$ is not NP-hard by showing the correctness condition for polynomial-time mapping reduction does not hold. An analogous argument shows that Σ^* is not NP-hard.
   > Let $L \in \mathsf{NP}$ be any non-empty language so there are elements of $x \in L$ and $x \notin L$. Any polytime mapping function f from L to $\emptyset$ must satisfy the correctness condition $x \in L \iff f(x) \in \emptyset$. By definition, for all y, $y \notin \emptyset$, so $x \in L$ cannot be mapped to some element of $\emptyset$.

   (b) If $\mathsf{P} = \mathsf{NP}$, then every language $L \in \mathsf{NP} \setminus \{\emptyset, \Sigma^*\}$ is NP-complete.

   > **Solution:** Fix $L \in \mathsf{NP} \setminus \{\emptyset, \Sigma^*\}$. Crucially, there exist some fixed (arbitrary) $y \in L$ and $\bar{y} \notin L$. Let $A \in \mathsf{NP} = \mathsf{P}$ and suppose A can be efficiently decided by D_A. We show $A \leq_p L$.
   > On input x:
   >
   > i. If $D_A(x)$ accepts (i.e. $x \in A$), return y.

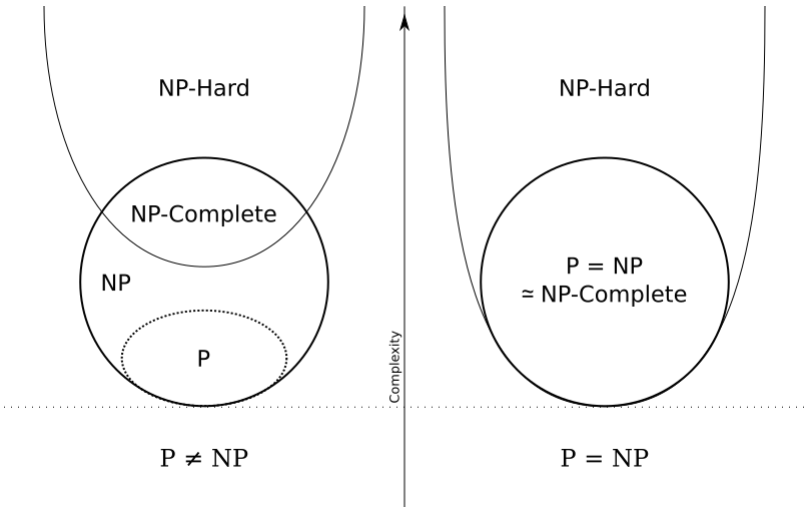
ii. If $D_A(x)$ rejects (i.e. $x \notin A$), return $\bar{y}$.

Runtime: The decider is efficient because D_A is an efficient.
Correctness: By construction, $f(x) \in L \iff D_A(x)$ accepts $\iff x \in A$.
Therefore, for all $A \in \mathsf{NP}$, $A \leq_p L$, so L is NP-hard.

3. Draw two different Venn diagrams which illustrate the relationships between the classes: NP, NP-Hard, NP-Complete and P in the case where:

   (a) $\mathsf{P} \neq \mathsf{NP}$
   (b) $\mathsf{P} = \mathsf{NP}$

Solution:



Note that if $P = NP$, then $P \approx \mathsf{NP}$-Complete rather than $P = \mathsf{NP}$-Complete. This is because regardless of whether P and NP are equal, $\varnothing, \Sigma^* \notin \mathsf{NP}$-Complete. L is NP-Hard iff $A \leq_p L$ for all $A \in \mathsf{NP}$ and $A \leq_p L$ iff there exists a polynomial time computable function $f\colon \Sigma^* \to \Sigma^*$ such that for every w, $w \in A \iff f(w) \in L$. Nevertheless, for any language $A \in NP$ where $A \neq \varnothing, \Sigma^*$, then there must exist at least one element that *is* in L and at least one element that is *not* in L so that "yes" instances in A can be mapped to "yes" instances in L and "no" instances in A can be mapped to "no" instances in L.

4. Show that L_{HALT} is NP-Hard.

Solution: We proceed by reduction from SAT. Notice that any NP-Hard language suffices to show this claim. The only differences in the proof will be the mapping. We have arbitrarily chosen to use SAT.

We need to generate a mapping $f\colon \Sigma^* \to \Sigma^*$ that has the following properties:

- $\phi \in \text{SAT} \implies f(\phi) \in L_{\text{HALT}}$

- $\phi \notin \text{SAT} \implies f(\phi) \notin L_{\text{HALT}}$
- f is polytime computable

where ϕ is a Boolean formula.

f outputs the pair (M_ϕ, ε), where M_ϕ is a Turing machine. M_ϕ behaves as follows on input x:

M_ϕ = "On input x:
1. Loop through all possible Boolean variable assignments a over the variables in ϕ.
2. If $\phi(a)$ is true, accept x.
3. If no assignment yields true, loop on x."

Notice that f outputs a Turing machine that is linear in size with respect to ϕ, so f is an efficient mapping.

Suppose $\phi \in \text{SAT}$. Then there is some variable assignment a such that $\phi(a)$ is true, and M eventually accepts ε, so $(M, \varepsilon) \in L_{\text{HALT}}$.

Suppose $\phi \notin \text{SAT}$. Then ϕ evaluated on *every* assignment a is false, and M eventually loops on ε, so $(M, \varepsilon) \notin L_{\text{HALT}}$.

Therefore $\text{SAT} \leq_p L_{\text{HALT}}$.

5. Consider the following language:

$$L_{\text{EQ}} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : L(M_1) = L(M_2)\}$$

Provide a reduction that proves L_{EQ} is NP-Hard.

Solution: We show $\text{SAT} \leq_p L_{\text{EQ}}$. Let A denote a Turing machine which always accepts its input.

```
1: function M(x)
2:     for each possible assignment a of φ do
3:         if φ(a) = 1 then
4:             Accept
5:     Loop
```

We map a Boolean formula ϕ to the machine pair M, A (where M is as defined above).

Clearly, A can be constructed efficiently as it does not depend on the input, and M can be constructed in time linear to the size of the formula ϕ. This implies that the reduction is efficient.

Suppose $\phi \in \text{SAT}$. Then there is some satisfying assignment for ϕ and eventually M accepts. As x is independent of the execution of M, we have that $L(M) = \Sigma^* = L(A)$ as desired.

Suppose $\phi \notin$ SAT. Then there is no satisfying assignment for ϕ and M loops. As x is independent of the execution of M, we have that $L(M) = \varnothing \neq \Sigma^* = L(A)$ as desired.

6. (True/ False/ Unknown) Let $L_{\text{LIS}} = \{(S, k) : S \text{ is a sequence of numbers, and } S \text{ has a strictly increasing subsequence of size greater than } k\}$. L_{LIS} is NP-Complete.

Solution: Unknown.
We can define a polynomial-time decider for this language. Recall the Longest Increasing Subsequence Dynamic Programming algorithm from Lecture 5. This algorithm operates on a sequence S with length n, memoizing to a memo with dimensions $1 \times n$. This is a bottom-up implementation of that algorithm:

```
1: function L(S, k)
2:     Initialize a table M with dimension 1 × n
3:     for i = 1 to n do M[i] ← 1
4:         for j = 1 to i − 1 do
5:             if S[i] > S[j] and M[j] ≥ M[i] then
6:                 M[i] ← M[j] + 1
7:             if M[i] > k then accept
8:     reject"
```

This algorithm runs the Longest Increasing Subsequence DP algorithm to find the longest increasing subsequence of S. If S, k is in the language then there exists some such subsequence with length greater than k; the decider will return true once it finds the last element in such a subsequence. If S, k is not in the language, there cannot exist an increasing subsequence with length greater than k; the longest increasing subsequence will have length at most k and so the decider will reject. This decider runs in polynomial time on the input length. In both the inner and outer loops, it iterates over the length of S at most; this is $O(|S|^2)$, and polynomial in the size of the input. The comparisons are either between numbers in the input, or numbers that are at most $\max(|S|, k)$ and therefore have size at most $O(\log |S| + \log k)$, so the comparisons are also efficient. Having constructed an efficient decider, we conclude this must be in P.

Since $L_{\text{LIS}} \in \mathsf{P}$, it is NP-Complete if and only if $\mathsf{P} = \mathsf{NP}$. This is unknown, so the correct answer is **unknown**.

7. An undirected graph has a *bicycle* if it has two non-intersecting cycles of equal size that together include all the vertices. We define BICYCLE $= \{\langle G \rangle : G \text{ has a bicycle}\}$.

   Prove that BICYCLE is NP-complete.

Solution: First, we show that BICYCLE is in NP, via the following efficient verifier. Given G and a certificate $C = (C_1, C_2)$ where (C_1, C_2) are the two purported cycles, we confirm that C_1 and C_2 together include every vertex in G exactly once, that they are of equal

size, and that they are both cycles. This can be done straightforwardly in polynomial time. Clearly, any $\langle G \rangle \in$ Bicycle has a certificate that will make this verifier accept, and only graphs with bicycles have such certificates.

Now we show that Bicycle is NP-hard, via a polynomial-time mapping reduction from the NP-complete HamCycle problem to Bicycle. The function, given G, just "doubles" it, outputting a new graph G' that consists of two copies of G (with no edges between the copies; G' does not have to be connected).

This reduction is clearly efficient. We now show that it is correct: if G has a Hamiltonian cycle, then G' clearly has a bicycle consisting of the same Hamiltonian cycle in its two copies of G, as required. Conversely, if G' has a bicycle, then because its two copies of G are not connected to each other in any way, the bicycle in G' must consist of a Hamiltonian cycle in each copy of G (it need not be the same cycle). Therefore, G has a Hamiltonian cycle, as required.

8. Prove that Clique $\leq_p$ IndependentSet, where

   Clique $= \{(G = (V, E), k) :$ There exists a subset of V of size k that is fully connected$\}$

   Recall that IndependentSet $= \{(G = (V, E), k) :$ There exists a subset of V of size k such that no two vertices share an edge$\}$.

Solution: In this proof, we will show that Clique $\leq_p$ IndependentSet.

$f(G = (V, E), k)$:

1) Create edge set $E' = \emptyset$
2) For each distinct pair of vertices (u, v):
3) If edge $(u, v) \notin E$, add (u, v) to E'
4) Set $G' = (V, E'), k' = k$
5) return G', k'

Line 2 iterates over every pair of vertices which is $O(|V|^2)$. Line 3 scans for an edge in E which is $O(|E|)$. All other work done is trivial. Since all steps taken in this function are polynomial with respect to the input size, this function is efficient.

Assume that $(G, k) \in$ Clique, i.e. there exists a clique of size k in G. This means that there is a set of k vertices that are fully connected. The output of f is a graph with the same vertices and value of k, but any edge that exists in the original graph will not exist in the new graph. If there will be no edges between our set of k vertices that originally comprised the clique, then what we're left with is an independent set of size k, which means that the output of the function is in the language IndependentSet.

Assume that $(G', k') \in$ IndependentSet, i.e. there exists an independent set I of size k' in G'. This means that there is not an edge between any pair of vertices in I. If there is

not an edge in G', that means that there must have been an edge in the original graph G between the same two vertices, as our function outputs the complement of the graph. If every non-edge in G' is an edge in G, this means that our subset I was fully connected in the original graph, so there was a clique of size $k' = k$. Thus, the input must have been in the language CLIQUE.

9. A *square vertex cover* is a vertex cover whose size is a perfect square, meaning that the size is k^2 for some integer k. Consider the following language:

$$\text{SQUAREVERTEXCOVER} = \left\{ \langle G, k \rangle : G \text{ has a vertex cover of size at most } k^2 \right\}.$$

Show that SQUAREVERTEXCOVER is NP-Complete.

Solution: First, we show that SQUAREVERTEXCOVER is in NP, via the following efficient verifier. Given $\langle G, k \rangle$ and a certificate $C = (v_1, \ldots, v_{k^2})$ where $v_1, \ldots, v_{k^2}$ is a purported vertex cover, we first verify that $|C| \leq k^2$. We then collect all the edges incident to the vertices in C and verify that this set is equal to all the edges in G. This can be done straightforwardly in polynomial time. Clearly, any $\langle G, k \rangle \in$ SQUAREVERTEXCOVER has a certificate that will make this verifier accept, and only graphs with vertex covers of size k^2 have such certificates.

Now we show that SQUAREVERTEXCOVER is NP-hard, via a polynomial-time mapping reduction from the NP-complete VERTEXCOVER problem to SQUAREVERTEXCOVER. The function, given $(G = (V, E), k)$ outputs an instance of SQUAREVERTEXCOVER as follows:

- If $k \leq |V|$, the function creates a new clique of size $n - k + 1$ for a perfect square $k \leq n \leq 2k$ (the distance between consecutive squares is on the order of $O(\sqrt{n})$, which you can see by comparing n^2 and $(n+1)^2$), outputting $(G', \sqrt{n})$, where G' is a new graph that consists of G plus the new clique (with no edges between the original vertices and the new ones).
- If $k > |V|$, the function just outputs $(G, \sqrt{n})$, where n is computed as above.

This reduction is efficient; it generates at most $O(|V|^2)$ new vertices and edges, which is efficient in the input size. We now show that it is correct.

- In the first case above, if G has a vertex cover of size k, then G' clearly has a vertex cover consisting of the the same k vertices plus all but one of the additional vertices, for a cover of size $k + n - k + 1 - 1 = n$, which is a perfect square. Conversely, if G' has a vertex cover of size n, then the cover must contain $n - k$ of the vertices in the newly added clique to cover the edges internal to that clique. The remaining k vertices in the cover must cover all the edges from the original graph G. Therefore, G has a vertex cover of size k, as required.
- In the second case above, G trivially has a vertex cover consisting of all the vertices, and this number is bounded above by both k and n. So both $(G, k) \in$ VERTEXCOVER and $(G, \sqrt{n}) \in$ SQUAREVERTEXCOVER in this case.

An alternate reduction would create G' as k copies of G, outputting (G', k), in the first case above. The analysis is similar to that above.

In both alternatives, the mapping function must be efficient in the input size, i.e. polynomial in $|G| + |k|$. Since there is no limitation on k with respect to $|G|$, the case where k is not polynomial with respect to $|G|$ must be handled carefully. However, we did not enforce this in grading this problem.