Practice Problems on NP-completeness
prepared by Steven Kwok

1. (a) Show that the 2-satisfiability problem is in P.
   (b) Show that if \( k \geq 4 \), the \( k \)-satisfiability problem is NP-complete.

2. Show that the Hamiltonian Path problem is NP-complete by reducing the Hamiltonian Cycle problem to it. (Assuming you know Hamiltonian Cycle is NP-complete but do not know whether Hamiltonian Path is NP-complete.)

3. Show that the Traveling Salesman Problem is NP-complete by reduction from Hamiltonian Cycle to Traveling Salesman Problem. (Assuming you know Hamiltonian Cycle is NP-complete but do not know whether Traveling Salesman Problem is NP-complete.)

4. Show that the following problem are NP-complete.
   (a) MINIMUM SUM OF SQUARES
   INSTANCE: Finite set \( A \), "size" \( s(a) \in \mathbb{Z}^+ \) for each \( a \in A \), positive integers \( K \) and \( J \).
   QUESTION: Can the elements of \( A \) be partitioned into \( K \) disjoint sets \( A_1, \ldots, A_k \) such that
   \[
   \sum_{i=1}^{k} \left( \sum_{a \in A_i} s(a) \right)^2 \leq J ?
   \]
   (b) FEEDBACK VERTEX SET
   INSTANCE: Directed Graph \( G = (V, A) \), positive integer \( K \leq |V| \).
   QUESTION: Is there a subset \( V' \subseteq V \) such that \( |V'| \leq K \) and such that every directed circuit in \( G \) includes at least one vertex from \( V' \)?
   (c) DOMINATING SET
   INSTANCE: Graph \( G = (V, E) \), positive integer \( K \leq |V| \).
   QUESTION: Is there a subset \( V' \subseteq V \) such that \( |V'| \leq K \) and such that every vertex \( v \in V - V' \) is joined to at least one member of \( V' \) by an edge in \( E \) ?

5. Page 960 Exercise 36.5-2

7. Each city has one snowplow. It must plow its narrow streets, which the plow can plow completely in one traversal. There are also county roads that the plow can traverse to get from one place to another, though there don’t need to be plowed. In other words, we have a weighted graph with edges of two types; those of type 1 must be traversed, those of type 2 need not be. The state wants an algorithm that will find a minimum-length tour traversing the type 1 edges in such a graph. Prove that this problem is NP-hard.

Hints and Solutions

1. (a) Let \( f \) be an arbitrary 2-CNF (an expression of conjunctive normal form in which every conjunct is the disjunction of exactly two literals) defined over the set of variables \( x_1, \ldots, x_n \).
   We design a simple\(^1\) polynomial time algorithm for deciding if \( f \) is satisfiable.
   We claim that it is sufficient to construct a polynomial-time algorithm that behaves as follows.
   - If \( f \) is satisfiable then \( \text{test}(f, x_n) \) returns YES only if the formula, \( f|_{x_n = \text{true}} \), obtained by assigning \( \text{true} \) to \( x_n \), is also satisfiable.
   - If \( f \) is unsatisfiable then \( \text{test}(f, x_n) \) can either returns YES or NO.
   This is because we can use \( \text{test}(f, x_n) \) to decide the 2-satisfiability problem as described below.

---

\(^1\)There is a more efficient algorithm but the proof of correctness is slightly longer. Here, we pick simplicity over efficiency.
We prove the correctness of the above algorithm by induction on the number of variables, assuming that \(\text{test}(f, x_n)\) exists. Clearly, the algorithm behaves as desired if there is only one variable. Suppose the algorithms behave as desired for formulas with up to \(n - 1\) variables. Let \(f\) be a 2-CNF defined over the set of variables \(x_1, \ldots, x_n\). Consider the following cases when we first run 2-satisfiability(\(f\)), i.e., \(i = n\).

1. Suppose \(f\) is unsatisfiable. \(\text{test}(f, x_n)\) of Line 2 can either return YES or NO. In both cases, the resulting \(f'\) obtained from setting \(f\) to true or false is still not satisfiable. Thus, we can replace \(f\) by either \(f|_{x_n = \text{true}}\) or \(f|_{x_n = \text{false}}\).

2. Suppose \(f\) has a truth assignment \(\theta\) such that \(\theta(x_n) = \text{true}\). Then \(f|_{x_n = \text{true}}\) is also satisfiable. Here the test \(\text{test}(f, x_n)\) in Line 2 returns YES and we substitute \(f\) by \(f|_{x_n = \text{true}}\) which is satisfiable.

3. Suppose all truth assignments set \(x_n\) to false. Then, \(f|_{x_n = \text{false}}\) is also satisfiable. Here, the test \(\text{test}(f, x_n)\) in Line 2 returns NO and \(f\) is replaced by \(f|_{x_n = \text{false}}\) which is satisfiable.

Thus, 2-satisfiability works correctly and it runs in polynomial time if the subroutine \(\text{test}(f, x_i)\) also runs in polynomial time. It remains to construct the subroutine \(\text{test}(f, x_i)\).

\[
\text{test}(f, x_i) \\
1. g = f|_{x_i = \text{true}} \\
2. \text{while } g \text{ contains a clause consisting of only a single literal } l \text{ do} \\
3. \quad g = g|_{l = \text{true}}^2 \\
4. \quad \text{if } g \text{ is the constant expression false then return NO and halt end if} \\
5. \quad \text{if } g \text{ is the constant expression true then return YES and halt end if} \\
6. \text{end while} \\
7. \text{return YES.}
\]

In the while loop, clearly, if \(g\) contains a clause consisting of only a single literal \(l\) then \(l\) has to be set to \(\text{true}\) in order that \(g\) is satisfiable. Hence, we can set \(l\) to true in \(g\) (Line 3) and observe the following.

- If the resulting \(g\) is false then the original formula \(g = f|_{x_i = \text{true}}\) cannot be satisfied (Line 4).
- If the resulting \(g\) is true then we have a truth assignment for the original \(g = f|_{x_i = \text{true}}\) (Line 5).
- If we reach line 7, the final formula \(g\) is a 2-CNF formula such that it is satisfiable if and only if the original formula \(g = f|_{x_i = \text{true}}\) is satisfiable. In this case, if \(f\) is satisfiable then it returns YES (at Line 7) only if \(f|_{x_i = \text{true}}\) is satisfiable, as desired.

The time complexity is clearly polynomial.

The same argument that shows satisfiability problem is in \(NP\) applies here (refer to text). Thus, it suffices to show that \(k\)-satisfiability, \(k \geq 4\) is \(NP\)-hard. This is done by reducing the \(NP\)-complete problem 3-SAT to the \(k\)-satisfiability (\(k \geq 4\)) problem as follows.

Let \(f = \bigwedge_{i=1}^{k} C_i\), where each clause \(C_i\) is a disjunction of exactly three literals \(l_{i,1}, l_{i,2}, l_{i,3}\), be an arbitrary instance of the 3-SAT problem. We replace each clause \(C_i\) by

\[
C_i' = l_{i,1} \lor l_{i,2} \lor l_{i,3} \lor \overset{k-3}{\overbrace{y \lor \ldots \lor y}}
\]

Let \(f' = (\bigwedge_{i=1}^{k} C_i') \land (y \lor \overset{k}{\overbrace{\ldots \lor y}})\). If \(\theta\) is a truth assignment for \(f\) then \(\theta\) together with \(y = \text{true}\) must also satisfy \(f'\). Conversely, to satisfy \(f'\), \(y\) must be set to \(\text{true}\) and the resulting formula (obtained by setting \(y\) to true in \(f'\)) is the same as \(f\). Thus, if \(f'\) is satisfiable then \(f\) is satisfiable. Clearly, the reduction takes polynomial time.
2. Hint: Introduce one more vertex.

3. Hint: Introduce weights to the edges.

4. (a) Hint: Reduce from Partition.
   (b) Hint: Reduce from Vertex Cover.
   (c) Hint: Reduce from Vertex Cover. Introduce a triangle for each edge.

5. We leave as an exercise for the reader to verify that 0-1 integer programming problem is in $NP$.

   To prove that 0-1 integer programming problem is $NP$-hard, we reduce 3-CNF-SAT to 0-1 integer programming problem. Let $v_1, \ldots, v_k$ be a set of variables. Suppose we have the following instance to the 3-CNF-SAT problem,

   \[ f = \land_{i=1}^{k} \lor_{j=1}^{3} l_{i,j} \]

   where $l_{i,j}$ is either $v_i$ or $\overline{v_i}$ for some index $i$.

   We construct a 0-1 integer program from $f$ as follows. We associate the variable $x_i$ to each positive literal $v_i$ and the variable $x'_i$ to each negative literal $\overline{v_i}$. In other words, $x_i = 1$ if $v_i = 1$ and $x'_i = 1$ if $v_i = 0$. For each clause $C_i = b_{i,1} \lor b_{i,2} \lor b_{i,3}$ in $f$, we introduce the inequalities

   \[ \left( \sum_{i=1}^{n} -a_{i,j} x_i - a'_{i,j} x'_i \right) \leq -1 \]

   where $a_{i,j}$ and $a'_{i,j}$ are the number of times $a_i$ and $\overline{a_i}$ appear in the clause $C_i$, respectively. We can then let $A = (a_{i,j})$, $x = (x_1, \ldots, x_n, x'_1, \ldots, x'_n)$ and $b = (-1, \ldots, -1)$.

   For example, if $f$ is the 3-CNF formula $(x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_3})$ then the 0-1 integer program is

   \[ \left( \begin{array}{cccccc}
   -1 & 0 & -1 & 0 & 0 & -1 \\
   0 & -1 & 0 & -2 & 0 & 0 \\
   \end{array} \right) \preceq \left( \begin{array}{c}
   x_1 \\
   x_2 \\
   x_3 \\
   x'_1 \\
   x'_2 \\
   x'_3 \\
   \end{array} \right) \]

   If there is a truth assignment $\theta$ that satisfies $f$ then the left-hand side of each inequality is at most -1 where the $x_i$'s and $x'_i$'s are set corresponding to $\theta$. Thus, if $f$ is satisfiable then there is a $2n$-vector $x$ that satisfies $Ax \leq b$. However, this reduction is not complete in the sense that the existence of a $2n$-vector that satisfies $Ax \leq b$ does not imply that $f$ is satisfiable.

   As a pathological example, consider $f = (v_1 \lor v_1 \lor v_1) \land (\overline{v_1} \lor \overline{v_1} \lor \overline{v_1})$ which is not satisfiable. However, its corresponding 0-1 integer program

   \[ \left( \begin{array}{cc}
   -3 & 0 \\
   0 & -3 \\
   \end{array} \right) \preceq \left( \begin{array}{c}
   x_1 \\
   x'_1 \\
   \end{array} \right) \]

   can be satisfied by assigning 1 to both $x_1$ and $x'_1$.

   The problem in the above counterexample is that we did not ensure, for each $i$, exactly one of the $x_i$ and $x'_i$ is set to one while the other is set to zero. To remedy this problem, we introduce the following two inequalities for each $i$.

   \begin{align*}
   x_i + x'_i & \leq 1 \\
   -x_i - x'_i & \leq -1
   \end{align*}

   The first inequality says that at most one of $x_i$ and $x'_i$ is set to one. While the second inequality says that at least one of $x_i$ and $x'_i$ is set to one. Together, they imply that exactly one of $x_i$ and $x'_i$ is set to 1.

   To see that the resulting 0-1 integer program is satisfiable implies that $f$ is satisfiable, simply note that

   (a) The second half of the reduction ensures that for each $i$, exactly one of $x_i$ and $x'_i$ is set to one, thus we can unambiguously set $v_i$ to true or false according to whether $x_i = 1$ or $x'_i = 1$.

   \footnote{The smallest reasonable example is more time consuming to construct.}
(b) Those inequalities, of the form
\[
\sum_{i=1}^{t} a_{i}^{t} x_{t} - a_{i}^{t} x_{t}' \leq -1,
\]
that are introduced in the first part of the reduction are satisfied implies that by instantiating the variables according to (a), each clause \( C_i \) is satisfied.

Clearly, the construction of the integer program can be done in time \( O(kn) \) where \( k \) is the number of clauses in \( f \). Thus, we have reduce 3-CNF-SAT to 0-1 integer programming, proving that 0-1 integer programming is NP-hard.

Hence 0-1 integer programming is NP-complete.

6. (a) Given an arbitrary graph \( G(V,E) \), we can determine whether it can be 2-colored by first constructing a breath-first search tree \( T(G) \) of \( G \) in polynomial time, as described in section 23.2 of CLR. If there is a non-tree edge of \( G \) that connects a pair of vertices \( u \) and \( v \) such that the difference in their depths is even then we know that there is an odd cycle in the graph \( G \) (we leave this as a simple exercise for the reader to verify). Since there is no way we can color an odd cycle with 2 colors, \( G \) cannot be 2-colored. On the other hand, if there is no such edge, then we can 2-color the vertices on the odd level the color 1 and on the even level the color 0.

(b) The graph-coloring problem as a decision problem:

**INSTANCE**: Given a graph \( G = (V,E) \) and an integer \( k \leq |V| \).

**QUESTION**: Is there a coloring \( c : V \rightarrow \{1, \ldots, k\} \) such that
\[
c(u) \neq c(v) \text{ for every edge } (u,v) \in E.
\]

If we can solve the graph-coloring problem in polynomial time, then we can solve the decision problem by first determining what is the minimum number of colors needed to color a graph and see if it is greater than \( k \).

Conversely, if we can solve our decision problem in polynomial time then we can solve the graph-coloring problem as follows.

1. for \( k = 1 \) to \( |V| \)
2. if \( G \) can be \( k \)-colored then
3. return \( k \) and halt
4. end if
5. end for

The time complexity of both cases (under their respective assumptions) are clearly polynomial.

(c) The decision problem is clearly in NP since we can nondeterministically guess a \( k \)-coloring of the graph and verify that our guess is indeed a \( k \)-coloring by checking that each adjacent pair of vertices are colored differently.

We can reduce the 3-COLOR problem easily to the decision problem in part (b) simply by setting \( k = 3 \). Hence, if the 3-COLOR problem is NP-complete then the decision problem in part (b) is NP-hard and thus, NP-complete.

(d) Since a variable and its negation are both connected to the vertex RED, they cannot be colored \( c(\text{RED}) \) and hence have to be colored either \( c(\text{TRUE}) \) or \( c(\text{FALSE}) \). Moreover, since a variable is connected to its negation, exactly one of them is colored \( c(\text{TRUE}) \) and the other is colored \( c(\text{FALSE}) \).

If a variable \( x_i \) is set to true in a truth assignment then we color \( x_i \) \( c(\text{TRUE}) \) and color its negation \( \overline{x_i} \) \( c(\text{FALSE}) \). The triangle \( x_i\overline{x_i}(\text{RED}) \) are properly 3-colored and the triangle \( (\text{TRUE})(\text{FALSE})(\text{RED}) \) are properly 3-colored by definition. The graph consisting of the literal edges consists of exactly these triangles and thus, can be 3-colored.

(e) For ease of explaining, let us label the vertices of Figure 36.20 as shown below.

By part (d), none of the vertices \( x, y \) and \( z \) can be colored \( c(\text{RED}) \). Thus, it suffices to show that there is no 3-coloring of the widget with all \( x, y \) and \( z \) receiving the color \( c(\text{FALSE}) \).

Suppose \( x, y \) and \( z \) are colored \( c(\text{FALSE}) \). Then vertices 3 and 4 have to be colored either \( c(\text{TRUE}) \) or \( c(\text{RED}) \) but not \( c(\text{FALSE}) \). Now, if vertex 2 is colored \( c(\text{RED}) \) then both vertices 3 and 4 have to
be colored c(TRUE) which is illegal since they are connected. Hence, vertex 2 has to be colored either c(TRUE) or c(FALSE). By the same argument, vertex 2 cannot be colored c(TRUE) (we leave it to the reader as a simple exercise).

Thus, we suppose vertex 2 is colored c(FALSE). Then vertex 1 being adjacent to both the vertices 2 and TRUE, has to be colored c(RED). This forces the vertex 5 to receive the color c(FALSE) since it is adjacent to both vertices 1 and TRUE. However, we have an illegal coloring here since both vertex 5 and z are adjacent and are colored c(FALSE).

(f) As in part (c), the 3-COLOR problem is clearly in NP since we can nondeterministically guess a 3-coloring of the graph and verify that our guess is indeed a 3-coloring by checking that each adjacent pair of vertices are colored differently.

To prove that 3-COLOR is NP-hard, we reduce 3-CNF-SAT to 3-COLOR. The reduction is as described in the problem (the paragraph before part (d)).

Suppose \( \phi \) is satisfiable by the truth assignment \( \theta \). We color the vertices \( x \) c(TRUE) and \( \bar{x} \) c(FALSE) if \( \theta(x) = \text{true} \). Otherwise, we color \( x \) c(FALSE) and \( \bar{x} \) c(TRUE). By part (d), this partial coloring forms a 3-coloring of the graph containing just the literal edges. Since \( \phi \) is satisfied, each clause has a literal that is set to true and hence the corresponding vertex receives the color c(TRUE). By part (e), the widget corresponding to this clause can be 3-colored.

Conversely, suppose the graph \( G \) is 3-colorable. We can assign the value true to a variable if its corresponding vertex in \( G \) gets color c(TRUE), otherwise we assign false to it. We assign value to its negation in the same manner and by part (d), such assignment is a valid one (i.e., we do not have a situation where we assign both \( x \) and \( \bar{x} \) the same value). Moreover, since the widget corresponding to each clause is 3-colorable, part (e) implies that each clause has a literal that is assigned true. Thus, \( \phi \) is satisfiable.

7. **Hint:** reduction from HAMILTONIAN CYCLE problem. Introduce an edge for each vertex.