# CS 577: Introduction to Algorithms NP-Completeness Review Solutions

## Solutions

1. First, we prove that NAE-SAT is in NP. Given an assignment of variables, we can check whether $\psi$ is satisfied such that each clause contains one literal that is 1 and one literal that is 0; this check is of course performed in time polynomial with respect to the number of variables and clauses. Thus, NAE-SAT is in NP.

   Next, we prove that NAE-SAT is NP-hard by showing that an instance of CNF-SAT can be reduced to an instance of NAE-SAT. Let $\varphi$ be the formula for an instance of CNF-SAT and $\psi$ be the formula formed by the reduction to an instance of NAE-SAT. These steps outline the reduction from CNF-SAT to NAE-SAT:

   - For $\psi$, define variables $y_i,\ i = 1, \ldots, n$ where $n$ is the number of variables in $\varphi$, and $z$.
   - For each clause $C_j$ in $\varphi$, create a clause $C'_j$ for $\psi$ by converting all variables $x_i$ in $C_j$ to $y_i$, then adding $z$ to the clause. For example, the clause $(x_1 \vee x_2 \vee x_3)$ in $\varphi$ becomes $(y_1, y_2, y_3, z)$ in $\psi$.

   $\psi$ can be created in polynomial time. Now we prove that $\psi$ has a satisfying assignment if and only if $\varphi$ has a satisfying assignment.

   Suppose $y^*$ and $z$ form a satisfying assignment for $\psi$. For $i = 1, \ldots, n$, let

   $$x_i^* = \begin{cases} 1, & y_i^* \neq z \\ 0, & y_i^* = z. \end{cases}$$

   Equivalently, we have

   $$\neg x_i^* = \begin{cases} 1, & \neg y_i^* \neq z \\ 0, & \neg y_i^* = z. \end{cases}$$

   Since $y^*$ and $z$ form a satisfying assignment for $\psi$, each clause $C'_j$ in $\psi$ contains a literal that is 1 and a literal that is 0. Furthermore, some $y_i^*$ must differ from $z$, thus some $x_i^*$ in the corresponding clause $C_j$ in $\varphi$ must be 1. Therefore, every clause $C_j$ in $\varphi$ is satisfied, so $\varphi$ is satisfied.

   Suppose $x^*$ is a satisfying assignment for $\varphi$. Consider any clause $C_j$ in $\varphi$; it must be the case that some literal is 1. Thus, we can form a satisfying assignment $y^*$ for $\psi$ by setting $y_i^* \leftarrow x_i^*,\ i = 1, \ldots, n$, and $z \leftarrow 0$. Every clause $C_j^*$ in $\psi$ will be satisfied because some literal will still be 1, and $z$ is 0, so not all literals will be equal. Therefore, $\psi$ is satisfied.

   Since CNF-SAT is NP-hard, NAE-SAT must also be NP-hard. We conclude that NAE-SAT is NP-complete.

2. First, we prove that MONOTONE-SAT is in NP. Given an assignment of variables, we can check whether $k$ or fewer variables are set to 1 and whether the function is satisfied in polynomial time. Thus, MONOTONE-SAT is in NP.

   Next, we prove that MONOTONE-SAT is NP-hard by showing a reduction from VERTEXCOVER to MONOTONE-SAT. Let $\mathcal{V}$ denote an instance of VERTEXCOVER, and $\phi$ denote the MONOTONE-SAT function created by the reduction. In addition, let $G$ denote the graph corresponding to $\mathcal{V}$. These steps outline the reduction:

   - For each vertex $v$ in $G$, create a variable $x_v$ in $\phi$.
   - For each edge $(u, v)$ in $G$, create a clause $(x_u \vee x_v)$ in $\phi$.

$\phi$ can be created in polynomial time with respect to the size of $G$. Now we prove that $\phi$ has a satisfying assignment if and only if $\mathcal{V}$ is a "Yes" instance.

Given a satisfying assignment for $\phi$, we can construct a vertex cover $S$ that satisfies $\mathcal{V}$ by including vertex $u$ in $S$ if and only if $x_u$ is set to 1 in the satisfying assignment for $\phi$. $S$ is a valid vertex cover because for each edge $(u, v)$, because of how $\phi$ was constructed in the reduction, either $x_u$, $x_v$, or both will be set to 1, thus at least one of $u$ or $v$ will be included in $S$. Furthermore, there is a one-to-one correspondence between the variables in $\phi$ and the vertices in $G$, so because $k$ or fewer variables are set to 1 in the satisfying assignment for $\phi$, $k$ or fewer vertices will be included in $S$. Therefore, we have a "Yes" instance for $\mathcal{V}$.

Given the vertex cover $S$ corresponding to a "Yes" instance for $\mathcal{V}$, we can construct a satisfying assignment for $\phi$ by setting variable $X_u$ to 1 if and only if vertex $u$ is included in $S$. $\phi$ will be satisfied because each clause $(x_u \vee x_v)$ directly corresponds to an edge $(u, v)$ in $G$, and since $S$ is a vertex cover of $G$, either $u$, $v$, or both are included in $S$, thus either $x_u$,, $x_v$, or both are set to 1. Again, since there is a one-to-one correspondence between the variables in $\phi$ and the vertices in $G$, and $k$ are fewer vertices are included in $S$, $k$ or fewer variables will be set to 1 in the satisfying assignment to $\phi$. Therefore, we have a satisfying assignment to $\phi$.

Since VERTEXCOVER is NP-hard, MONOTONE-SAT must also be NP-hard. We conclude that MONOTONE-SAT is NP-complete.

3. First, we prove that MAXCUT is in NP. Given a proposed cut $(V_1, V_2)$, we verify that the total cost over all edges $(v_1, v_2)$ such that $v_1 \in V_1$ and $v_2 \in V_2$ is at least $k$.

   Next, we prove that MAXCUT is NP-hard. We will introduce a new problem called UNWEIGHTEDMAXCUT: Given a graph $G$ and a number $k$, determine whether $G$ has a cut $(V_1, V_2)$ with at least $k$ edges crossing the cut. Clearly, UNWEIGHTEDMAXCUT reduces to MAXCUT (set all edge weights to 1; rigorous proof omitted). Thus, if we show that NAE-3-SAT reduces to UNWEIGHTEDMAXCUT, we will have shown that NAE-3-SAT reduces to MAXCUT, thus proving that MAXCUT is NP-hard (an underlying assumption is that NAE-3-SAT is NP-hard; the proof for this is omitted).

   Roughly speaking, we will reduce NAE-3-SAT to UNWEIGHTEDMAXCUT by creating a graph such that the max cut partitions the graph into vertices on one side that correspond to literals that have value 1, and vertices on the other side that correspond to literals that have value 0. Let $x_1, \ldots, x_n$ be the $n$ variables in the NAE-3-SAT FUNCTION $\psi$ with $m$ clauses, and $G = (V, E)$ be the graph associated with the MAXCUT instance. These steps outline the reduction:

   - Add $2n$ vertices to $G$, namely $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$.
   - For each clause $(l_1, l_2, l_3)$ in $\psi$, we add edges $(l_1, l_2)$, $(l_2, l_3)$, and $(l_3, l_1)$ to $G$, thus adding $3m$ edges to $G$.
   - For each variable $x_i$ in $\psi$, add an edge from $x_i$ to $\neg x_i$ in $G$, thus adding $n$ edges to $G$.
   - Now, we have $3m + n$ edges in $G$. Let $k = 2m + n$.

   Clearly, $G$ can be constructed in polynomial time. Now we prove that this reduction is correct.

   Suppose $\psi$ has a satisfying assignment. Then the max cut partition is as follows: $V_1$ contains all vertices corresponding to variables in $\psi$ assigned 1, and $V_2$ contains all the other vertices. The cut will have one edge for each variable in $\psi$, as $x_i$ and $\neg x_i$ cannot both be in $v_1$ or $V_2$. In addition, for each clause, two literals will lie on one side of the cut, and one on the other. Thus, two edges will cross the cut for each clause. Therefore, the total number of edges crossing the cut is $2m + n$, which is the same as $k$.

   Conversely, suppose $G$ has a cut of size $k = 2m + n$. For each variable $x_i$, either both $x_i$ and $\neg x_i$ lie on the same side of the cut, or they lie on opposite sides. Thus, a variable contributes at most one edge to the cut. For a clause, either all literals lie on one side of the cut, or two lie on one and one on the other. Thus, a clause contributes at most two edges to the cut. Therefore, if there is a solution to MAXCUT, then every variable contributes an edge and every clause contributes two edges. Furthermore, we can use this solution to construct a satisfying assignment for $\psi$.

Since NAE-3-SAT is NP-complete, MAXCUT must also be NP-complete.

4. We can reduce HAMILTONIANPATH to LONGESTSIMPLEPATH by setting all edge costs to 1 and $k$ to $|V| - 1$; if there is indeed a longest simple path of length $k$, then we conclude all vertices were reached. Since the HAMILTONIANPATH is NP-complete, LONGESTSIMPLEPATH must also be NP-complete.

5. First, we prove that SHORTESTSIMPLEPATH is in NP. Given a proposed simple path, we can verify whether the path costs at most $k$. Thus, SHORTESTSIMPLEPATH is in NP.

   Next, we prove that SHORTESTSIMPLEPATH is NP-hard by showing a reduction from LONGESTSIMPLEPATH to SHORTESTSIMPLEPATH. We reduce an instance of Longest Simple Path $\mathcal{L}$ to an instance of SHORTESTSIMPLEPATH $\mathcal{S}$ by negating all edge weights in $G$, the graph corresponding to $\mathcal{L}$, to obtain $G'$, a graph we will associate to $\mathcal{S}$. We then determine whether $G'$ has a simple path of cost at least $-k$.

   Since LONGESTSIMPLEPATH is NP-hard, SHORTESTSIMPLEPATH must also be NP-hard. We conclude that SHORTESTSIMPLEPATH is NP-complete.

   *Note:* If the edge costs are nonnegative, Dijkstra's algorithm will solve SHORTESTSIMPLEPATH in polynomial time. Equivalently, we can solve LONGESTSIMPLEPATH in polynomial time if the edge costs are nonpositive. Recall the Bellman-Ford algorithm, which appears to be a valid solution to SHORTESTSIMPLEPATH. The reason we cannot use Bellman-Ford here is in the case of negative-cost cycles; Bellman-Ford will simply report the existence of such a cycle and terminate, thus it is an unacceptable algorithm for LONGESTSIMPLEPATH.

6. (a) Another term for a graph being 2-colorable is that it is bipartite.
   A *Bipartite Graph* is a graph whose vertices can be divided into two independent sets, $U$ and $V$ such that every edge $(u, v)$ either connects a vertex from $U$ to $V$ or a vertex from $V$ to $U$. In other words, for every edge $(u, v)$, either $u$ belongs to $U$ and $v$ to $V$, or $u$ belongs to $V$ and $v$ to $U$. We can also say that there is no edge that connects vertices of same set.

   Following is a simple algorithm to find out whether a given graph is Bipartite or not using Breadth First Search (BFS).

   i. Assign RED color to the source vertex - randomly chosen source vertex (putting into set $U$).
   ii. Color all the neighbors with BLUE color (putting into set $V$).
   iii. Color all neighbor's neighbor with RED color (putting into set $U$).
   iv. This way, assign color to all vertices such that it satisfies all the constraints of $m$-way coloring problem where $m = 2$.
   v. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite)

   (b) 2-SAT is a special case of Boolean Satisfiability Problem and can be solved in polynomial time.
   Now, 2-SAT limits the problem of SAT to only those Boolean formula which are expressed as a CNF with every clause having only 2 terms (also called 2-CNF). For the CNF value to come TRUE, value of every clause should be TRUE. Let one of the clause be (A $\vee$ B).
   (A $\vee$ B) = TRUE

   - If A = 0, B must be 1 i.e. ($\bar{A} \Rightarrow B$)
   - If B = 0, A must be 1 i.e. ($\bar{B} \Rightarrow A$)

   Thus, (A $\vee$ B) = TRUE is equivalent to ($\bar{A} \Rightarrow B$) $\wedge$ ($\bar{B} \Rightarrow A$) Now, we can express the CNF as an Implication. So, we create an Implication Graph which has 2 edges for every clause of the CNF. (A $\vee$ B) is expressed in Implication Graph as edge($\bar{A} \rightarrow B$) and edge($\bar{B} \rightarrow A$).
   Thus, for a Boolean formula with 'm' clauses, we make an Implication Graph with:

   - 2 edges for every clause i.e. '2m' edges.
   - 1 node for every Boolean variable involved in the Boolean formula.

Now, consider the following cases:

- CASE 1: If edge($X \rightarrow \bar{X}$) exists in the graph This means ($X \Rightarrow \bar{X}$). If X = TRUE, $\bar{X}$ = TRUE, which is a contradiction. But if X = FALSE, there are no implication constraints. Thus, X = FALSE
- CASE 2: If edge($\bar{X} \rightarrow X$) exists in the graph This means ($\bar{X} \Rightarrow X$). If $\bar{X}$ = TRUE, X = TRUE, which is a contradiction. But if $\bar{X}$ = FALSE, there are no implication constraints. Thus, $\bar{X}$ = FALSE i.e. X = TRUE
- CASE 3: If edge($X \rightarrow \bar{X}$) and edge($\bar{X} \rightarrow X$) both exist in the graph. One edge requires X to be TRUE and the other one requires X to be FALSE. Thus, there is no possible assignment in such a case.

If any two variables X and $\bar{X}$ are on a cycle i.e. path($\bar{A} \rightarrow B$) and path($\bar{B} \rightarrow A$) both exists, then the CNF is unsatisfiable. Otherwise, there is a possible assignment and the CNF is satisfiable. Note here that, we use path due to the following property of implication: If we have ($A \Rightarrow B$) and ($B \Rightarrow C$), then $A \Rightarrow C$ Thus, if we have a path in the Implication Graph, that is pretty much same as having a direct edge.

7. We will reduce from CNF-SAT. Given a CNF-SAT formula $\varphi = C_1 \wedge \ldots \wedge C_m$ with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$, we can construct an instance of the solitaire problem $\mathcal{S}$ so that $\mathcal{S}$ has a solution if and only if $\varphi$ has a satisfying assignment. These steps outline the reduction:

   - Create an $m \times n$ solitaire grid $A$. Each row $i$ will correspond to clause $C_i$, and each column $j$ will correspond to variable $x_j$.
   - For each clause $C_i$,
     - If $x_j$ is present in nonnegated form (as $x_j$), put a blue stone in $A[i][j]$.
     - If $x_j$ is present in negated form (as $\neg x_j$), put a red stone in $A[i][j]$.

   *Note:* The reduction assumes that $x_j$ and $\neg x_j$ cannot both appear within the same clause. If they are, then because $x_j \vee \neg x_j = 1$, discarding a clause containing $x_j \vee \neg x_j$ will have no impact on $\varphi$.

   We prove that $\mathcal{S}$ has a solution if and only if $\varphi$ has a satisfying assignment.

   Suppose that $x_j^*,\ j = 1, \ldots, n$ is a satisfying assignment to $\varphi$. Upon constructing $\mathcal{S}$, we remove red stones from column $j$ if $x_j^* = 1$, or blue stones otherwise ($x_j^* = 0$). This leaves every column with one type of stone in it. We now verify that every row has at least one stone in it. Consider a row $i$. Since $x_j^*,\ j = 1, \ldots, n$ is a satisfying assignment to $\varphi$, some literal in $C_i$ is set to 1, thus there must exist some $j$ such that the stone in the $i$th row and $j$th column is left alone. Therefore, every row of $\mathcal{S}$ has a stone, hence $\mathcal{S}$ is solvable.

   Now suppose that $\mathcal{S}$ is solvable. This means that every column contains only one type of stone (or has no stones in it, in which case we associate some arbitrary stone type to it). Moreover, for each row, there is a stone remaining in that row. This solution to $\mathcal{S}$ can be used to construct a satisfying assignment $x_j^*,\ j = 1, \ldots, n$ to $\varphi$. For each column $j$, if the column contains only blue stones, set $x_j^* := 1$, else (column contains only red stones) set $x_j^* := 0$ (or set $x_j^*$ arbitrarily if the column contains no stones). For any clause $C_i$, we know that in the solution to $\mathcal{S}$, there is some stone in some column $j$ in the $i$th row. This corresponds to the literal set to 1. Thus, every clause in $\varphi$ is satisfied, and so $\varphi$ is satisfied. This concludes the proof.

8. For each vertex $v$ in $G = (V, E)$, associate to $v$ a decision variable $x_v$. The following ILP is a reduction of VERTEXCOVER:

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x \geq 0 \\ & x \leq 1 \\ & x_u + x_v \geq 1,\ \forall (u, v) \in E. \end{aligned}$$

   For the decision variant of VERTEXCOVER with target $k$, an instance is a "Yes" instance if there exists some feasible $x$ such that $\sum_{v \in V} x_v \leq k$.

9. Suppose the graph $G$ defines an instance of 3-COLORING. For each vertex $v$, define a variable $x_v$, and define a clause $C_v = (x_{v^(1)}, \ldots, x_{v^(k)})$, where $x_{v^(1)}, \ldots, x_{v^(k)}$ are the nodes adjacent to $v$ (note that $k$ must be less than 3, otherwise we know right away that $G$ is not 3-colorable). Let $v_1, \ldots, v_n$ be the set of nodes in $G$, and $\phi = C_{v_1} \wedge \ldots \wedge C_{v_n}$; $\phi$ is the formula for an instance of 1-IN-3 SAT.

10. We reduce from the Partition problem: Given a set of positive integers $S$, is it possible to partition the numbers into two sets $S_1$ and $S_2$ such that the sum over all numbers in $S_1$ equals the sum over all numbers in $S_2$? These steps outline the reduction from an instance $\mathcal{P}$ to an instance of RectangleTiling $\mathcal{R}$:

    - Sum all the numbers in $S$. Assign this value to $a$; an underlying assumption is that $a$ is an even number, so if this is not the case, then we immediately know that $\mathcal{P}$ is a "No" instance.
    - Assign to $b$ any integer greater than $\frac{a}{2}$.
    - Create a large rectangle $A$ of dimensions $2b \times \frac{a}{2}$.
    - For each integer $i$ in $S$, create a rectangle $B_i$ of dimensions $b \times i$.

    $A$ and $B_i,\ i \in S$, form an instance of $\mathcal{R}$.

11. We begin with a mapping reduction from VERTEXCOVER to SETCOVER. Given an instance $(G, k)$ of VERTEXCOVER, the reduced instance of SETCOVER is defined as follows:

    - The universe $U$ is the set of edges, $E$, of $G$.
    - The set family $\mathcal{P}$ contains the sets $S_v$ for every vertex $v$ in $G$, where $S_v$ is the set of edges incident to $v$, i.e., $S_v = \{e \in E \mid v \text{ is an end point of } e\}$.
    - The threshold is the same as the threshold $k$ of the VERTEXCOVER instance.

    We now prove that this is in fact a reduction.

    **Claim:** *$G$ has a vertex cover of size $\leq k$ if and only if $U$ is covered by less than $k$ sets in $\mathcal{P}$.*

    *Proof of Claim:*

    $\Rightarrow$ We first show that if $G$ has a vertex cover of size $\leq k$ then $U$ has a set cover of size $\leq k$ sets in $\mathcal{P}$. Let $C = \{v_1, \ldots, v_l\}$ be a vertex cover of $G$ of size $l$, for some $l \leq k$. Every edge in the graph is incident to at least one of these vertices in $C$. In other words every element of $U$ lies in at least one of the sets $S_{v_1}, \ldots, S_{v_l}$. Therefore, $U$ has a set cover of size $l \leq k$ in $\mathcal{P}$.

    $\Leftarrow$ For the other direction suppose there is set cover $\mathcal{S}$ of $U$ in $\mathcal{P}$ of size $l(\leq k)$. Let $C$ denote the set of vertices that correspond to the sets in $\mathcal{S}$. Since every element of $U$ lies in at least one of the sets in $\mathcal{S}$, every edge in the graph is incident to one of the vertices in $C$. Therefore $G$ has a vertex cover of size $l(\leq k)$.

12. This problem deals with the following versions of VERTEXCOVER.

    (A) Decision VERTEXCOVER:

    Input A pair $(G, k)$, where $G$ is a graph and $k$ positive integer.
    Output 'YES' if $G$ has a vertex cover of size $\leq k$, 'NO' otherwise.

    (B) Search VERTEXCOVER:

    Input A pair $(G, k)$, where $G$ is a graph and $k$ positive integer.
    Output A vertex cover of size $\leq k$ if one exists, and 'NO' otherwise.

    (C) Optimization version of VERTEXCOVER:

    Input A graph $G$.
    Output A vertex cover of minimum size.

(a) We establish a poly-time reduction from (C) to (B). Let $ALG_B$ be an algorithm for Search VERTEX-COVER. We give an algorithm for the optimization version of VERTEXCOVER that runs in polynomial time and makes a polynomial number of function calls to $ALG_B$. The idea is to run $ALG_B$ on $G$ with smaller and smaller values of $k$, till we obtain the smallest $k$ for which there is a vertex cover of size $k$. $ALG_B$ is then used to find such a vertex cover. Below is a formal description of this algorithm

Input: Graph $G$ on $n$ vertices.

Output: Vertex cover of minimum size.

$k \leftarrow n-1$.
**While** ($ALG_B(G,k) \neq$ 'NO')
  $k \leftarrow k-1$
**End**
**Return** $ALG_B(G,k+1)$

(b) We now give a poly-time reduction from (B) to (A). Let $ALG_A$ be an algorithm for the decision version of VERTEXCOVER. The key observation in obtaining an algorithm for the search version is that we can use function calls to $ALG_A$ to determine whether a given vertex $v$ in $G$ is a member of a vertex cover of size less than or equal to $k$. To do so delete all edges incident to $v$ and check if the resulting graph contains a vertex cover of size $k-1$ using $ALG_A$. If it does then $v$ is part of a vertex cover of $G$ of size $\leq k$ and if not, there is no vertex cover of size $k$ containing $v$. By repeated application of this idea, we obtain the following algorithm for the problem.

Input: Graph $G$ on $n$ vertices, threshold $k$.

Output: Vertex cover $C$ of size $k$ if one exists, 'NO' otherwise.

**If** ($ALG_A(G,k) =$ 'NO') **Return** 'No'.
$S \leftarrow \{\}$
**For** each vertex $v$ in $G$
  Delete all edges incident to $v$ to obtain a graph $G'$.
  **If** ($ALG_A(G',k-1) =$ 'Yes')
    $G \leftarrow G'$
    $k \leftarrow k-1$
    Add $v$ to $S$
  **EndIf**
  **If** $k=0$ **Return** $S$
**EndFor**