Exercise 10: More NP-Completeness and Beyond

1. The Knapsack problem is defined as follows. We have n items, each with positive integral *weight* w_j $(j = 1, \cdots, n)$ and positive integral *value* c_j $(j = 1, \cdots, n)$ and an integer b. The question is to find a subset of the items with total weight at most b and maximal value.

   Answer the following questions:

   (a) Give the decision version of the knapsack problem, KNAPSACK.

   There are two ways to present the definition:

   - Given n items, items, each with positive integral weight w_j $(j = 1, \cdots, n)$ and positive integral value c_j $(j = 1, \cdots, n)$ and an integer b. Is there a set of items where the total weight is at most b and the total cost at least k?
   - KNAPSACK $= \{\langle W, C, b, k\rangle |$ There are n items. The weight of items are positive integers $W = \{w_1, \cdots, w_n\}$. The value of items are positive integral $C = \{c_1, \cdots, c_n\}$. And b is positive integral capacity. There is a set of items where the total weight is at most b and the total cost is at least k.$\}$

   (b) Prove that KNAPSACK problem is NP-complete.

   First, we prove that KNAPSACK is in NP. Let a set of items with a total weight of at most b and a total value of at least k be a certificate C. The verifier first checks if the items in C are the items in the input. Next, it checks if the total weight is at most b and the total value is at least k. The checking takes $O(n \cdot \min\{|C|, n\} + |C| + |C|)$ time. That is, it is verifiable in polynomial time in the input length.

   Next, we prove that KNAPSACK is NP-hard by polynomial-time reduction from PARTITION. For any instance of PARTITION, $S = \{x_1, \cdots, x_n\}$, we transform it into an instance of KNAPSACK, (W, C, b, k) as follows.

   - $W = S = \{x_1, \cdots, x_n\}$
   - $C = S = \{x_1, \cdots, x_n\}$
   - $b = \frac{\Sigma_i x_i}{2}$
   - $k = \frac{\Sigma_i x_i}{2}$

   The transformation can be done in polynomial time as it only copies the input and sums up the numbers.

   Next, we prove that there is a partition of set S if and only if there is a set of items such that the total weight is at most b and the total value is at least k. Assume that there is an equal-sum partition of S, one of the parties has total value $\frac{\Sigma_i x_i}{2}$. The corresponding items in the KNAPSACK problem have total weight $\frac{\Sigma_i x_i}{2} \leq b$ and total value $\frac{\Sigma_i x_i}{2} \geq k$.

   Assume that there is a set of items S' such that the total weight is at most b and the total value is at least k. That is, the items in S' has total weight $\leq \frac{\Sigma_i x_i}{2}$ and total cost $\geq \frac{\Sigma_i x_i}{2}$. Since for any item, its weight is equal to its value, the total weight is equal to the total cost and is equal to $\frac{\Sigma_i x_i}{2}$. Therefore, the items that are not in S' have total weight (and total value) equal to $\frac{\Sigma_i x_i}{2}$, too. Therefore, the subsets S' and $S \setminus S'$ are a partition of S.

2. Given a graph $G = (V, E)$, a *vertex cover* is a subset C of vertices in V such that for any edge $(u, v) \in E$, $\{u, v\} \cap C \geq 1$. In the *Minimum Vertex Cover* problem, we aim at finding the minimum vertex cover in the given graph.

(a) Give the decision version of the Minimum Vertex Cover, VC

VC = Given a graph G, is there a vertex cover in G with size at most k?
(Alternative: VC = $\{\langle G, k\rangle \mid$ There is a vertex cover in G with size at most $k\}$)

(b) Show that the decision version of the Minimum Vertex Cover, VC, is NP-complete.

Proof. First we prove that VC is in NP. Let a vertex cover C of size k be the certificate. The verifier first checks if C has less than or equal to k vertices. Next, the verifier checks for every edge that if it has at least one endpoint in C. The verification takes linear time ($O(\min\{|C|, |V|\})$) for checking the size of C. For checking if the elements in C are in V, it takes $O(|V|^2)$ time. The final checking takes $O(|E| \cdot |D|)$ time. Hence, the verification can be done in polynomial time in the input length of $G = (V, E)$.

Next, we show that VC is NP-hard by Polynomial time reduction from IndependentSet. Given an instance of the IndependentSet problem, $G = (V, E)$ and integer k, we construct an instance of the VC, G' and integer k' where $G' = G$ and $k' = |V| - k$. The reduction takes polynomial time since G' is a copy of G and k' can be calculated in constant time.

Now we want to show that the reduction works by proving that there is a independent set in G of size at least k if and only if there is a vertex cover in G' of size at most k'.

Suppose that G has an independent set $I \subseteq V$ with $|I| \geq k$. We claim that $V \setminus I$ is a vertex cover in G'. Since I is an independent set in G, for any edge $(u, v) \in E$, u and v cannot be both in I. That is, one of u or v is in $V \setminus I$. Hence, the set of vertices $V \setminus I$ is a vertex cover, which has size $|V| - |I| \leq |V| - k$.

Suppose that there is a vertex cover C in G' where $|C| \leq |V| - k$. For all edge $(u, v) \in E'$, at least one of u or v is in C. That is, for any pair of u and v which are both not in C, $(u, v) \notin C$. Therefore, the set $V \setminus C$ forms an independent set, which has size $|V| - |C| \geq k$. □

3. Consider the maximum weighted vertex cover problem: given a graph $G = (V, E)$ and each vertex $v \in V$ has weight $w(v)$, find a vertex cover with minimum weight.

Answer the following questions:

(a) Give the decision version of the minimum vertex cover problem, WeightedVC.

There are two ways to present the definition:

- Given a weighted graph G and number k, is there a vertex cover in G with total weight at most k?
- Weighted-VC = $\{\langle G, k\rangle |$There is a vertex cover in G with total weight $\leq k\}$.

(b) Prove that WeightedVC is NP-hard.

We first show that Weighted-VC is in NP. We prove it by showing that it is polynomial verifiable. Using a vertex cover C as the certificate, the verifier checks if C is a subset of V, checks if every edge $(u, v) \in E$ has at least one endpoint in C, and checks if the total weight of the vertices in U is no more than k. Since $|C| \leq |V|$, this needs at most $O(|V|^2 + |V||E| + |V|)$ time, which is polynomial to the size of G.

Next we show that Weighted-VC is NP-complete by reduction from VertexCover = $\{\langle G, k\rangle |$There is a vertex cover in G with size $\leq s\}$. For any instance of VertexCover, we construct an instance of Weighted-VC, $G' = G = (V, E)$ and $k = s$. For the weight of the vertices, we set $w(v) = 1$ for all $v \in V$. This construction can be done in polynomial time since we only duplicate the input graph and set up the weight for each vertex once.

Now we show that the reduction works. That is, G has a vertex cover of size $\leq s$ if and only if G' has a weighted vertex cover with total weight $\leq k$.

Suppose there is a vertex cover U in G with size at most s, the same set of vertices in G' is also a vertex cover since $E' = E$. The total weight of the set of vertices is $|U| \leq s = k$. Hence, it is a vertex cover which has total weight at most k.

Conversely, suppose there is a vertex cover U in G' with total weight at most k. The set of vertices in U is a vertex cover in G since $E = E'$. The size of U is $|U| \leq k = s$ as $w(v) = 1$ for any vertex $v \in V'$. Hence, it is a vertex cover which size is at most s.

4. Recall that in the load balancing problem, there are m machines and n jobs, where job i has processing time p_i. The load of a machine is the total processing time of the jobs assigned to it. The goal is to assign the jobs on the machines such that the highest load among the machines is minimized.

   Show that the load balancing problem is NP-complete.

   First, we define the decision version of load balancing problem.

   $$\textsc{LoadBalancing} = \{\langle P, m, k\rangle \mid P = \{p_1, p_2, \cdots, p_n\} \text{ where } p_i \text{ is the processing time of job } i.$$
   $$\text{All the jobs can be assigned to } m \text{ machines}$$
   $$\text{such that the highest load among the } m \text{ machines is at most } k.\}$$

   Next, we show that LoadBalancing is in NP by proving that it is polynomial-time verifiable. Given a certificate that is a collection of subsets $\{S_1, S_2, \cdots, S_m\}$. The verifier first checks if $S_1 \cup S_2 \cup \cdots \cup S_m$ is equal to P. Then it finds the subset with the largest sum $S^* = \max_i \sum_{p \in S_i} p$ and checks if the sum of S^* is at most k, i.e., $\sum_{p \in S^*} p \leq k$. The verifier runs in $O(n^2 + n + n)$ time, which is polynomial to n.

   Finally, we show that LoadBalancing is NP-hard by a reduction from Partition. Given an instance $A = \{a_1, a_2, \cdots, a_n\}$ of Partition, we construct an instance of LoadBalancing as follows:

   - $P = A$.
   - $m = 2$.
   - $k = \frac{\sum_{a \in A} a}{2}$.

   We show that A has a partition iff P can be assigned to m machines with load no more than k. If A has a partition, then P can be divided into two subsets with equal sum. Assigning the two subsets to the two machines gives a schedule with machine load no more than k. If P can be assigned to $m = 2$ machines with load no more than $k = \frac{\sum_{a \in A} a}{2}$, then each of the two machines must have the load equal to $\frac{\sum_{a \in A} a}{2}$. Otherwise, one of the machines has load less than $\frac{\sum_{a \in A} a}{2}$ and the sum of the two machine loads is less than the sum of P, which is a contradiction. Therefore, the two machines constitute a partition of S. The construction of P, m and k runs in $O(n + 1 + n)$ time, which is polynomial to n.

5. Consider the *machine minimization* problem as follows. There are n jobs $J_1, J_2, \cdots, J_n$. Each job J_i has *processing time* p_i and *feasible interval* $I_i = [r_i, d_i]$ where J_i should be assigned. There are unlimited number of machines. Each machine can execute at most one job at a time. A *feasible* schedule is an assignment of every job J_i to a machine M_j at certain time t_i such that $[t_i, t_i + p_i] \subseteq [r_i, d_i]$, and there is no other jobs J_k assigned to the same machine such that $[t_k, t_k + p_k] \cap [t_i, t_i + p_i] \neq \phi$.

   The decision version of the machine minimization problem is

   $$\textsc{MachineMinimization} = \{\langle S, P, L, k\rangle \mid S = \{J_1, J_2, \cdots, J_n\}, P = \{p_1, p_2, \cdots, p_n\}, \text{ and}$$
   $$L = \{I_1, I_2, \cdots, I_n\}.\ S \text{ is a set of jobs where each job } J_i \text{ has processing time } p_i$$
   $$\text{and feasible interval } I_i.\ \text{The jobs in } S \text{ can be feasibly scheduled on at most } k \text{ machines.}\}$$

   Show that MachineMinimization is NP-hard.

Proof. We prove that MachineMinimization is NP-hard by polynomial-time reduction from Partition.

For any instance of Partition, $S = \{x_1, x_2, \cdots, x_n\}$, we transform it to an instance of MachineMinimization, (S', P, L, k) as follows:

- $S' = \{J_1, J_2, \cdots, J_{|S|}\}$.
- $P = S$. That is, $p_i = x_i$ for all $1 \leq i \leq n$.
- For any $I_i \in L$, $I_i = [0, \frac{\Sigma_i x_i}{2}]$.
- $k = 2$.

This transformation can be done in polynomial time.

Now, we show that if there is a feasible schedule of the MachineMinimization problem instance, the corresponding instance of Partition has a equal-sum partition. If there is a feasible schedule of $(S', P, L, k$, it means that the jobs can be scheduled on $k = 2$ machines. Also, since the total processing time is $\Sigma_i x_i$ and each feasible interval is of length $\frac{\Sigma_i x_i}{2}$, the total processing time of jobs assigned to one machine is exactly $\frac{\Sigma_i x_i}{2}$. Hence, the sum of numbers x_i corresponding to jobs assigned to one machine is $\frac{\Sigma_i x_i}{2}$. Therefore, there is a partition in S.

Next, we show that if there is a partition of S, the corresponding instance of MachineMinimization has a feasible schedule. If there is a partition of S, the elements in S can be partitioned into two parties with equal sums $\frac{\Sigma_i x_i}{2}$. Consider the jobs corresponding to the number of one of the parties. The total processing time of these jobs is $\frac{\Sigma_i x_i}{2}$. Since the jobs have uniform feasible intervals with length $\frac{\Sigma_i x_i}{2}$, they can be feasibly scheduled on one machine. Therefore, there is a feasible schedule using 2 machines for the instance (S', P, L, k). □

6. Given a graph $G = (V, E)$, a *dominating set* is a set of vertices $D \subseteq V$ such that for any vertices $v \in V$, $v \in D$ or at least one of the neighbors of v is in D. The goal of the Minimum Dominating Set problem is to find a dominating set of the given graph with the minimum cardinality.

   Show that the Minimum Dominating Set problem is NP-complete.

   First, we define the decision version of Minimum Dominating Set problem.

$$\text{DominatingSet} = \{\langle V, E, k\rangle \mid \text{There is a subset } D \subseteq V \text{ such that for all } v \in V$$

$$v \in D \text{ or } u \in D \text{ where } (v, u) \in E, \text{ and } |D| \leq k.\}$$

   Next, we show that DominatingSet is in NP by proving that it is polynomial-time verifiable. Given a certificate C of a vertex set, the verifier first checks if C is a subset of V and $|C| \leq k$. Then, for all $v \in V$, it checks if v is in C or one of the neighbors of v is in C. The verifier runs in $O(\min\{|C|, |V|\}|V| + (|V| + |E|)\min\{|C|, |V|\})$ time, which is polynomial to $|V|$ and $|E|$.

   Finally, we show that DominatingSet is NP-hard by a reduction from the decision version of Minimum Vertex Cover. Given an instance $\langle V, E, k\rangle$ of the decision version of Minimum Vertex Cover, we construct an instance $\langle V', E', k'\rangle$ of DominatingSet as follows:

   - $V' = V \cup U'$ where there is a vertex $u_{x,y} \in U'$ for each edge $(x, y) \in E$.
   - $E' = E \cup F'$ where there are two edges $(u_{x,y}, x) \in F'$ and $(u_{x,y}, y) \in F'$ for each $u_{x,y} \in U'$.
   - $k' = k + s$ where s is the number of isolated vertices in V.

   We show that $\langle V, E\rangle$ has a vertex cover of size at most k iff $\langle V', E'\rangle$ has a dominating set of size at most k'.

   If $\langle V, E\rangle$ has a vertex cover C of size at most k, then C must also dominate V and U'. Since C covers E, C dominates all non-isolated vertices in V. For each edge $(x, y) \in E$, C covers (x, y), $(u_{x,y}, x) \in F'$ and $(u_{x,y}, y) \in F'$, so C dominates $u_{x,y} \in U'$. Lastly, we use s additional vertices

to dominate all the isolated vertices. Thus, C and the s vertices are a dominating set with size at most k' for $\langle V', E' \rangle$.

If $\langle V', E' \rangle$ has a dominating set D of size at most k', then we can change D to be a vertex cover of $\langle V, E \rangle$. If D contains no vertices in U', by removing all isolated vertices in D, the result is a vertex cover of size at most k. Otherwise, there is some U' in D. For each $u_{x,y}$ in D, we remove $u_{x,y}$ from D and add x to D. This does not change the size of D. Since x, y and $u_{x,y}$ are the only vertices dominated by $u_{x,y}$ and they form a triangle, x must also dominates the three vertices. In this way, we can change D to another dominating set with the same size but contains no U'.

The construction of $\langle V', E', k' \rangle$ runs in $O((|V| + |E|) + |E| + |V|)$ time, which is polynomial to $|V|$ and $|E|$.

7. The EMPTYTM language is defined as $\{\langle M \rangle \mid M \text{ is a Turing machine and } M \text{ accepts nothing}\}$. Show that EMPTYTM is undecidable.

   Recall that ATM is the language

   $$\{\langle M, w \rangle \mid \text{Turing machine } M \text{ accepts string } w\}$$

   and ATM is undecidable. Suppose that EMPTYTM is decidable. Then there exists a decider D_E for EMPTYTM. We show in the following paragraphs that we can use D_E to construct a decider D_A that decides ATM. Therefore, we obtain a contradiction and EMPTYTM is undecidable.

   Since D_A is a decider for ATM, the input of D_A is $\langle M, w \rangle$ where M is a Turing machine and w is a string. We use M and w to construct a Turing machine M^* as follows.

   M^* takes a string x as the input:

   1. If $x \neq w$, reject.

   2. If $x = w$, run M on w, and accept if M accepts w.

   The language L^* defined by M^* is $L \cap \{w\}$ where L is the language defined by M. If $w \in L$, then $L^* = \{w\}$. If $w \notin L$, then $L^* = \phi$. We use this property to construct D_A.

   Given a decider D_E for EMPTYTM, we construct a decider D_A for ATM as follows.

   D_A takes Turing machine M and string w as the input:

   1. Use M and w to construct M^*.

   2. Run D_E on the encoding of M^*.

   3. If D_E accepts, reject.

   4. If D_E rejects, accept.

   Therefore, if EMPTYTM is decidable, ATM is also decidable, which contradicts to the fact that ATM is undecidable. Thus, EMPTYTM is undecidable.