Exercise 10: NP-Completeness and Optimization

1. Given a graph $G = (V, E)$, an *independent set* is a subset U of vertices in V such that there is no edge between any two vertices in U. In the *Maximum Independent Set* problem, we aim at finding the maximum independent set in the given graph.

   (a) Give the decision version of the Maximum Independent Set, IndependentSet

   IndependentSet = Given a graph G, is there an independent set in G with size at least k?

   (Alternative: IndependentSet = $\{\langle G, k\rangle \mid$ There is an independent set in G with size at least $k\}$)

   (b) Show that the decision version of the Maximum Independent Set is NP-complete.

   First we prove that IndependentSet is in NP. Let an independent set of size k, c, be a certificate.

   A = "On input $\langle\langle G, k\rangle, c\rangle$:

   1. Test whether c is a set of k nodes in G.
   2. Test whether G there is no edge between any pair of vertices in c
   3. If both 1 and 2 pass, *accept*; otherwise, *reject*."

   Step 1 takes at most $|c| = k$ times of scanning through the input. Step 2 takes at most $|c|^2$ times of scanning through the input. Hence, A runs in polynomial time in the input length.

   Now, we prove that IndependentSet is NP-hard by polynomial-time reduction from Clique. Given an instance $G = (V, E)$ and k of Clique problem, we construct a graph $G' = (V', E')$ and an integer k' for INDEP-SET as follows. We set $V' = V$ and $k' = k$. That is, For any two vertices u and v, we have $(u, v) \in E'$ if and only if $(u, v) \notin E$. The construction can be done in polynomial time.

   Now we show that there is a clique in G with size at least k if and only if there is an independent set with size at least k' in G'. Assume that W' is a subset of V' that is an independent set with size at least k. By the construction, any two vertices in W' are adjacent in G (since they are not adjacent in G'). Thus, the vertices in W' is a clique of G with size at least k.

   Assume that W is a subset of V that is a clique with size at least k. By the construction, any two vertices in W are not adjacent in G' (since they are not adjacent in G). Thus, the vertices in W is an independent set of G with size at least $k = k'$.

2. Given a graph $G = (V, E)$, a *vertex cover* is a subset C of vertices in V such that for any edge $(u, v) \in E$, $\{u, v\} \cap U \geq 1$. 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 $= \{\langle G, k\rangle | \text{There is a vertex cover in } G \text{ with total weight} \leq k\}$.

   (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 $= \{\langle G, k\rangle | \text{There is a vertex cover in } G \text{ with size} \leq s\}$. 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. 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.

   In this question you may use the fact that the following problems are NP-complete: PARTITION, SUBSETSUM, MACHINEMINIMIZATION, CLIQUE, INDEPENDENTSET, VERTEXCOVER. 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 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 total weight at most b and total value 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' has total weight (and total value) equal to $\frac{\Sigma_i x_i}{2}$, too. Therefore, the subsets S' and $S \setminus S'$ is a partition of S.

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

$$\text{MACHINEMINIMIZATION} = \{\langle S, P, L, k\rangle \mid S = \{J_1, J_2, \cdots, J_n\}, P = \{p_1, p_2, \cdots, p_n\}, \text{ and}$$

$L = \{I_1, I_2, \cdots, I_n\}$. S is a set of jobs where each job J_i has processing time p_i and feasible interval I_i. The jobs in S can be feasibly scheduled on at most k machines.}

Next, we prove that MachineMinimization is NP-hard by polynomial-time reduction from Partition. *MachineMinimization* is NP-hard.

Proof. 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). □