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

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

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

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

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.

    (b) Prove that WeightedVC is NP-hard.

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.

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

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

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

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