Homework Assignment Week 9: NP-Complete Problems

Hyuntaek Oh

ohhyun@oregonstate.edu

CS 514_400 Algorithm

Dec. 2, 2024

1. **In 0-1 Integer programming, there are a set of inequalities of the form: $a_{i,1}X_1 + \cdots + a_{i,n}X_n \geq b_i$, where $1 \leq i \leq m$, $a_{i,1}, \ldots, a_{i,n}$ and b_i are rational numbers and $X_1, \ldots, X_n$ are Boolean variables. The problem has a Yes answer if there are solutions to the Boolean variables that satisfy all inequalities. Show that the 0-1 integer programming is NP-complete.**

According to the introduction to Algorithms CLRS, to prove the 0-1 integer programming is NP-complete, we need to show that it is in NP, and then that SAT can be expressed as a 0-1 Integer programming.

1) The problem is in NP

The time complexity of the given inequalities is $O(mn)$ since each inequality involves a constant time computation for n, and there are m inequalities, which is polynomial in the size of the input. This means that 0-1 integer programming is in NP.

2) SAT can be expressed as a 0-1 Integer programming.

Any instance of SAT (Boolean satisfiable problem) can be reduced to an instance of 0-1 integer programming since Boolean variables in SAT $(x_1, x_2, \ldots, x_n)$ can be mapped to $X_1, X_2, \ldots, X_n$ in 0-1 integer programming, where $X_i \in \{1, 0\}$ correspond to $x_i \in \{True, False\}$. Moreover, each clause in SAT can be converted to an inequality in 0-1 integer programming. For instance, a clause $\neg x_1 + \neg x_2 + x_3$ is satisfied if at least one literal is true and can be converted to:

$$(1 - X_1) + (1 - X_2) + X_3 \geq 1$$

In this form of inequalities, $(1 - X_1)$ represents $\neg x_1$ because $X_1 = 1$ corresponds to $x_1 = True$, and $X_1 = 0$ corresponds to $x_1 = False$. This conversion can repeat for all clauses m inequalities $a_{i,1}X_1 + \cdots + a_{i,n}X_n \geq b_i$, where the coefficient $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$ and b_i are determined based on the literals in each clause.

Thus, the SAT instance is satisfiable if and only if the corresponding 0-1 Integer Programming instance has a solution.

2. **[Clique] Given an undirected graph G and an integer p, we want to determine if it has a clique, i.e., a subgraph where there is an edge between each pair of nodes, of size p. Show that the clique problem is NP-complete by a reduction from independent set.**

1) CLIQUE ∈ NP

We need to determine whether there exists a subset of vertices $V' \subseteq V$ such that $|V'| = p$ and every pair of vertices in V' is connected by an edge. A clique of size p is a subset of vertices V' such that all pairs in V' are connected. Given a solution V' is provided, we can verify that all pairs in V' are connected in polynomial time by checking the edges for all pairs, $\frac{p(p-1)}{2}$, meaning that $O(p^2)$ time. Thus, clique is in NP.

2) A reduction from independent set to clique: NP-hard

We start with an instance of the independent set problem, a graph $G = (V, E)$ and integer k, and then construct a new graph $G' = (V, E')$. The graph G' has edges E' such that $(u, v) \in E'$ if and only if $(u, v) \notin E$, meaning that the graph G' is a complement graph of G.

A subset of vertices V' forms an independent set of size k in G if and only if V' forms a clique of size k in G' since no two vertices in V' are connected in G, meaning that they are connected in the complement graph G'.

A solution to the clique problem on G' with $p = k$. If a clique of size k exists in G', it corresponds to an independent set of size k in G. Constructing the complement graph G' takes $O(|V|^2)$, which is polynomial time. Therefore, the reduction from independent set to clique is polynomial, meaning that it is in NP-hard.

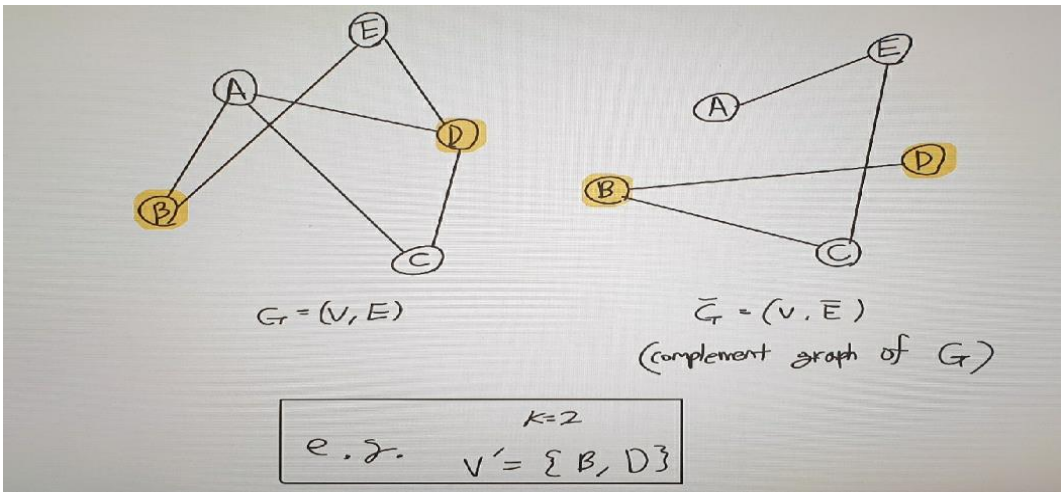

Fig. 1 The graph G and G'

Thus, we conclude that the clique problem is NP-complete.

3. [dHamPath] Show that determining if a directed graph has a directed Hamiltonian path, i.e., a directed path from some nodes s to some other node t that visits every other node exactly once is NP-complete by a reduction from dHamCycle.

1) Hamiltonian path is in NP

Hamiltonian path is that it is possible to traverse on all vertices of a directed graph $G = (V, E)$ without visiting one of the vertices more than once throughout the journey. The time complexity of Hamiltonian path is dependent on the path's length and edge existence, $O(|V|)$. Since the time complexity is related to the number of nodes, which is polynomial time, dHamPath is in NP.

2) A reduction from dHamCycle to Hamiltonian path: NP-hard

Assume a new graph $G' = (V', E')$. The vertex V' in G' consists of vertex V in G and two nodes s and t, $V' = V \cup \{s, t\}$. The edges E' in G' include E and additional edges (s, v_1) and (v_n, t), $E' = E \cup \{(s, v_1), (v_n, t)\}$. G has a directed Hamiltonian cycle if and only if G' has a directed Hamiltonian path from s to t.

Let's suppose that G has a Hamiltonian cycle $C = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow v_1$. In G', a Hamiltonian path is $P = s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow t$ where the path visits every vertex in V' exactly once, starting from s to t. From the path P, we can remove two nodes s and t, and then the remaining nodes can form Hamiltonian cycle by adding the edge (v_n, v_1), which exists in G.

Constructing G' takes $O(|V|)$ time, which is polynomial time, due to adding vertices and edges. Therefore, the reduction is polynomial, and the problem is NP-hard.

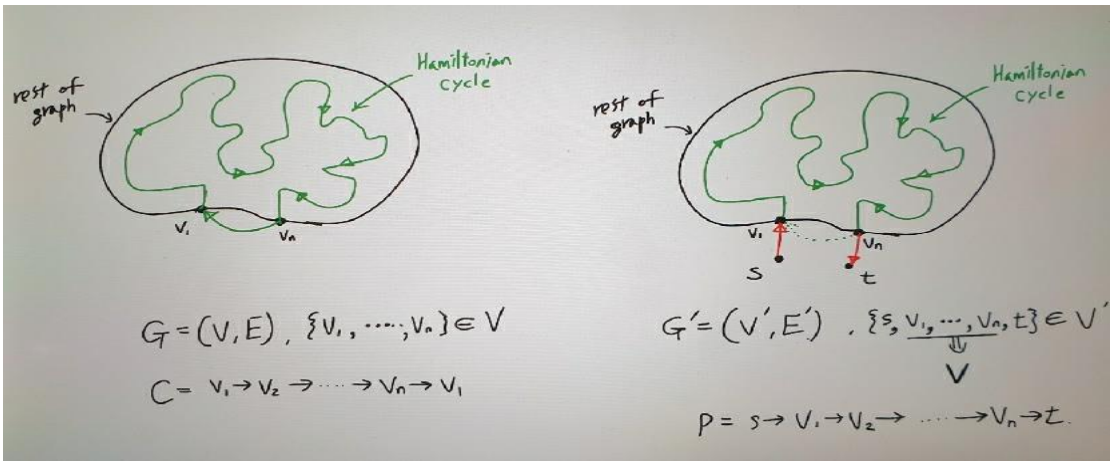

Fig. 2 dHamCycle and dHamPath

As a result, we can conclude that dHamPath is NP-complete.

4. **Given a set of n items of values $V_1, \ldots, V_n$ and weights $W_1, \ldots, W_n$, and a capacity W, the 0-1 Knapsack problem selects a subset of the items which can fit in the knapsack to maximize their total value. A decision version of this problem asks if there is selection of items which fit into the knapsack and has a value at least T. Reduce the subset sum problem to the knapsack problem to show that it is NP-hard.**

1) The Knapsack problem is in NP

Knapsack problem is to find the maximum total values with a capacity limit. It takes polynomial time to add the weights and profits of all possible items. The time complexity of the problem is $O(n * W)$ with bottom-up approach in dynamic programming. Thus, the knapsack problem is in NP.

2) Reduction from the subset problem to the knapsack problem: NP-hard

Let's assume that a set of integers $S = \{s_1, s_2, \ldots, s_n\}$ and a target sum T for subset problem. Given a set of n items, each with value and weight, and a maximum capacity, we can map an instance of the Subset sum problem to an instance of the Knapsack problem by setting $V_i = W_i = s_i$, which means that each item's value and weight are set to the corresponding element in the Subset sum problem. Knapsack capacity can also be setting $W = T$, meaning that the total weight capacity of the Knapsack is set to the target sum of the Subset sum problem.

If there exists the sum of a subset in the Subset sum problem, which is same with T, choosing these items in the 0-1 Knapsack setting will produce a total weight and value of T. This means that they fit in the Knapsack and meet the required value. On the other hand, if there exists an instance of a subset in the Knapsack that meets the weight and value requirements, the items' total weight and value will both be identical to T. This indicates that the subset of integers forms a valid solution to the Subset sum problem.

Thus, reduction from the Subset sum problem to Knapsack problem is NP-hard

5. Assume that there are n tasks with integer processing times $t_1, \dots, t_n$ that should be scheduled on two machines. Every task can be scheduled on either machine but not both. We want to minimize the total time by which all tasks are completed. Ideally the tasks can be scheduled so that the total processing time is equally divided. Show that determining if the processing time can be equally divided is NP-complete by reducing the subset sum problem to it.

1) The time scheduling is in NP

To verify the time scheduling problem is in NP, we need to check in polynomial time whether a solution satisfies the condition of equal processing time between the given two machines. The total processing time T can be computed:

$$T = \sum_{i=1}^{n} t_i$$

If T is $\left(\frac{T}{2}\right)$, equal division will be possible. For the given assignment of tasks to the two machines, which are T_1 and T_2 respectively, they should be:

$$T_1 = T_2 = \left(\frac{T}{2}\right)$$

Simple summations and comparisons, which are polynomial in the number of tasks n, are involved in the steps above. Therefore, the problem is verifiable in polynomial time and is in NP.

2) Reduction from the subset sum problem to the time scheduling problem: NP-complete

The subset $S = \{x_1, x_2, \dots, x_n\}$ in the subset sum problem can be mapped to the processing time of tasks $t_1, t_2, \dots, t_n$. The target value K in the subset sum problem can also correspond to $\left(\frac{T}{2}\right)$ in the time scheduling problem, where $T = \sum_{i=1}^{n} t_i$ is the total processing time. Then, we need to check a subset of S, which sums to K, can be reduced to the task distribution for one machine working $\left(\frac{T}{2}\right)$.

If there is a solution to the subset sum problem, there exists a subset of tasks whose total processing time equals $T/2$. Two machines are assigned tasks evenly, and the total processing time for both machines will be equal. On the other hand, if the time scheduling problem has a solution where $T_1 = T_2 = T/2$, the subset of tasks assigned to one machine forms a subset of S where the sum of it is equal to K.

The summation the task times and setting the target $T/2$ takes $O(n)$ time and constant time respectively. Thus, the reduction is polynomial, meaning that the time scheduling problem is NP-complete.