Assignment 9

CS 514 – Algorithms

Submitted By: Aman Pandita

Onid: panditaa@oregonstate.edu

1. (20 points) In **0-1 Integer programming**, there are a set of inequalities of the form: $a_{i,1}X_1 + \ldots + a_{i,n}X_n \geq b_i$, 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 an Yes answer if there are solutions to the Boolean variables that satisfy all inequalities. Show that the 0-1 Integer programming is NP-complete.

Hint: First show that it is in NP. Then show that Sat can be expressed as a 0-1 Integer program. This is one of the easier ways to reduce a problem to another. To reduce A to B, show that B can express A as a special case.

Answer 1:

To show that 0-1 integer programming is NP-complete, we need to demonstrate two things:

1. The problem is in NP

A problem is in NP if a solution to the problem can be verified in polynomial time.

For the 0-1 integer programming problem, given a set of Boolean variables as a candidate solution, we can verify whether this solution satisfies all the inequalities in polynomial time by simply substituting the values of $X_1,\ldots,X_n$ and checking each inequality.

The number of operations required is proportional to the number of inequalities and variables, which is polynomial with respect to the size of the input.

2. The problem is NP-hard

To show that 0-1 integer programming is NP-hard, we can reduce a known NP-complete problem to it. The Satisfiability problem (SAT) is a well-known NP-complete problem.

We can transform any instance of SAT into an instance of 0-1 integer programming as follows:

Given a SAT problem with clauses $C_1,\ldots,C_m$ over Boolean variables $Y_1,\ldots,Y_m$, we can construct a set of inequalities. Each clause C_j can be represented as an inequality by transforming each literal in the clause to a term in the inequality:

- If the literal is a positive occurrence of a variable Y_i, we use the variable X_i in the term.
- If the literal is a negated occurrence of a variable Y_i, we use $1 - X_i$ in the term.

For instance, if C_j is $(Y_1 \vee \neg Y_2 \vee Y_3)$, we can represent this clause by the inequality:

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

This ensures that the inequality is satisfied if and only if the clause is satisfied.

Each clause becomes an inequality, and since a solution to the SAT problem is an assignment of variables that satisfies all clauses, a solution to the constructed set of inequalities is an assignment of Boolean variables that satisfies all inequalities.

If we can find an assignment for $X_1,\ldots,X_n$ that satisfies all the inequalities derived from the SAT problem, then the original SAT problem is satisfiable.

Conversely, if the SAT problem is satisfiable, then there exists an assignment that satisfies the corresponding set of inequalities in the 0-1 integer program.

This shows a polynomial-time reduction from SAT to 0-1 integer programming, thus proving that 0-1 integer programming is NP-hard.

Conclusion
Since we have shown that 0-1 integer programming is in NP and that it is NP-hard by reducing SAT to it, we can conclude that 0-1 integer programming is NP-complete.

2. (20 points) **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.

Answer 2:

To show that the Clique problem is NP-complete, we will follow a similar two-step approach as before:

1. Prove the problem is in NP.

2. Show the problem is NP-hard by reducing a known NP-complete problem to it, in this case, the Independent Set problem.

1. The problem is in NP

A problem is in NP if any proposed solution can be verified quickly in polynomial time.
For the Clique problem, given a subset of nodes, we can check if every pair of nodes within this subset has an edge between them in polynomial time.
This verification process takes at most p_2 edge checks, which is $O(p^2)$ and thus polynomial in the size of the input.

2. The problem is NP-hard (Reduction from Independent Set)

The Independent Set problem is known to be NP-complete. In the Independent Set problem, we are given a graph ***G*** and an integer ***k***, and we need to determine whether there is a set of ***k*** mutually non-adjacent nodes in ***G***.

Here is how we can reduce the Independent Set problem to the Clique problem:

Given an instance of the Independent Set problem, with graph ***G*** and integer ***k***, we construct the complement graph $\bar{G}$ of ***G***. In $\bar{G}$, two nodes are connected with an edge if and only if they are not connected in ***G***.

Now, any independent set of size ***k*** in ***G*** corresponds to a clique of size ***k*** in $\bar{G}$ because if no two nodes in the set are adjacent in ***G***, all nodes in the set are mutually adjacent in $\bar{G}$.

Therefore, we can transform any instance of the Independent Set problem into an instance of the Clique problem. If we can find a clique of ***k*** in $\bar{G}$, then there is an independent set of size ***k*** in ***G***, and vice versa.

This transformation can be done in polynomial time because we can construct $\bar{G}$ by examining each possible pair of nodes in ***G*** and flipping the existence of an edge. This requires checking ***O(n^2)*** pairs for a graph with n nodes, which is polynomial time.

Conclusion

Since we have established that the Clique problem is in NP and have shown that any instance of the Independent Set problem (which is NP-complete) can be polynomially reduced to an instance of the Clique problem, the Clique problem is NP-hard.
Therefore, the Clique problem is NP-complete.

3. (20 points) **dHamPath:** Show that determining if a directed graph has a directed Hamiltonian path, i.e., a directed path from some node **s** to some other node **t** that visits every other node exactly once is NP-complete by a reduction from dHamCycle.

Answer 3:

To show that the Directed Hamiltonian Path problem (***dHamPath***) is NP-complete, we will again follow the two-step approach:
1. Prove the problem is in NP.
2. Demonstrate that the problem is NP-hard by reducing from the Directed Hamiltonian Cycle problem (dHamCycle), which is known to be NP-complete.

1. The problem is in NP
The Directed Hamiltonian Path problem is in NP because, given a directed graph and a proposed sequence of nodes, we can verify whether this sequence is a Hamiltonian path in polynomial time by checking the following:
- Each node appears exactly once in the sequence, except for the start and end if they are the same (for a cycle).
- There is a directed edge from each node to the next node in the sequence.

This verification involves checking ***n*** nodes and ***n***-1 edges, where ***n*** is the number of nodes in the graph, which can be done in polynomial time.

2. The problem is NP-hard (Reduction from dHamCycle)

To show that ***dHamPath*** is NP-hard, we will reduce the Directed Hamiltonian Cycle problem to it. The Directed Hamiltonian Cycle problem asks whether there is a directed cycle that visits every node exactly once in the graph.

Given an instance of the Directed Hamiltonian Cycle problem with a directed graph ***G***, we can construct an instance of the Directed Hamiltonian Path problem as follows:

1. Choose an arbitrary edge ***u, v*** in the graph ***G.***
2. Remove this edge to create a new graph ***G'***.
3. Now, ask whether there is a directed Hamiltonian path from ***u*** to ***v*** in ***G'***.

If the original graph ***G*** contains a Hamiltonian cycle, then by removing one edge from this cycle, we create a Hamiltonian path in ***G'*** from ***u*** to ***v***. Conversely, if ***G' has*** a Hamiltonian path from ***u*** to ***v***, adding the edge ***u, v*** back creates a Hamiltonian cycle in ***G***.

This reduction is polynomial in time since it only involves the removal of a single edge and does not depend on the size of the graph in any super-polynomial way.

Conclusion

Since ***dHamPath*** is in NP, and we can reduce ***dHamCycle*** (which is NP-complete) to ***dHamPath*** in polynomial time, it follows that ***dHamPath*** is also NP-hard. Therefore, the Directed Hamiltonian Path problem is NP-complete.

4. (20 points) Given a set of n items of values V_1, ...,V_n and weights W_1,...,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.

Answer 4:

To show that the 0-1 Knapsack problem is NP-hard, we will reduce the Subset Sum problem, which is known to be NP-hard, to it.

The Subset Sum Problem

The Subset Sum problem is defined as follows: given a set of integers $S = s_1, s_2, ..., s_n$ and a target sum ***T*** , determine if there is a subset of ***S*** whose sum is exactly ***T***.

Reduction to the 0-1 Knapsack Problem

The reduction can be done by transforming an instance of the Subset Sum problem into an instance of the 0-1 Knapsack problem in the following way:

- We take the set of integers $\boldsymbol{S = s_1, s_2, ..., s_n}$ from the Subset Sum problem and create ***n*** items for the Knapsack problem, where each item ***i*** has a value $\boldsymbol{V_i = s_i}$ and a weight $\boldsymbol{W_i = s_i}$.

- We set the capacity of the knapsack ***W*** to the target sum ***T*** from the Subset Sum problem.

- We set the minimum value that we want to achieve also to ***T***.

This transformation is done in polynomial time since it is a simple one-to-one mapping from the elements of the Subset Sum problem to the elements of the Knapsack problem.

Solving the Transformed Problem

Now, if there is a subset of ***S*** that sums to ***T***, then there will be a selection of items for the Knapsack problem that fits into the knapsack (since their total weight will be ***T***, which is the capacity of the knapsack) and has a value of at least ***T*** (since the value of each item is equal to its weight, and we are summing to ***T***.

Conversely, if there is a solution to the Knapsack problem where the selected items have a total value of at least ***T***, then the corresponding subset of integers ***S*** will sum to ***T***, because the value and weight are the same for each item.

Conclusion

Since we can transform any instance of the Subset Sum problem into an instance of the 0-1 Knapsack problem in polynomial time, and a solution to this instance of the Knapsack problem corresponds directly to a solution to the original Subset Sum problem, the 0-1 Knapsack problem is at least as hard as the Subset Sum problem. Therefore, the 0-1 Knapsack problem is NP-hard.

5. (20 points) Assume that there are n tasks with integer processing times t_1.. 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.

Answer 5:

The Subset Sum Problem

The Subset Sum problem is defined as follows: given a set of integers $S = s_1, s_2, ..., s_n$ and a target sum T, determine if there is a subset of S whose sum is exactly T.

Reduction to the Scheduling Problem

To reduce the Subset Sum problem to our scheduling problem, we consider the set of tasks $T = t_1, t_2, ..., t_n$ to be the same as the set S in the Subset Sum problem, with processing times equal to the integers in S.
We are then tasked with determining if it is possible to schedule these tasks on two machines such that the total processing time is equally divided between them.

If we can find a subset of S that sums to T, then we can schedule these tasks on one machine and the rest on the other machine, such that the total processing time for both machines is equal. This would be half of the sum of all processing times, which is the sum of all integers in the set S.

Conversely, if we can schedule the tasks between two machines such that the processing times are equal, then the set of tasks scheduled on one machine provides a subset of S whose sum is T, which solves the Subset Sum problem.
The transformation is polynomial in time, as it involves assigning each task a processing time equal to the corresponding integer in the Subset Sum problem and checking if a perfect partition exists.

Conclusion

Since we can transform any instance of the Subset Sum problem into an instance of our scheduling problem in polynomial time, and solving the scheduling problem would also solve the Subset Sum problem, our scheduling problem is at least as hard as the Subset Sum problem. Given that the Subset Sum problem is known to be NP-complete, it follows that our scheduling problem is also NP-hard.

Additionally, the scheduling problem is in NP because given a particular schedule, we can verify in polynomial time whether it results in an equal division of processing time between the two machines. Therefore, the scheduling problem is NP-complete.