CS 514

Assignment 9 – NP Complete Problems and Approximation Algorithm

report9.txt

Q1. (20 points) In 0-1 Integer programming, there are a set of inequalities of the form: $a_{i,1} X_1 + \ldots + a_{i,n} X_n >= b_i$ where $1 >= i >= m$, $a_{i,1}, \ldots, a_{i,n}$ 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.

Solution –

To establish the NP-completeness of 0-1 Integer Programming, we need to address two key aspects:

1. **Verification in NP:**

We illustrate that 0-1 Integer Programming is in NP by efficiently verifying the validity of a proposed solution, where Boolean values are assigned to variables, satisfying all given inequalities.
The verification process involves confirming that assigned Boolean values adhere to the binary nature (0 or 1) and that these values satisfy the specified inequalities, represented as follows:

$\sum^{n}_{j=1} a_{i,j} X_j >= b_i$

Here, X_j represents the Boolean variables, $a_{i,j}$ and b_i are rational numbers, and $1 <= i <= m$.

Both verification steps can be completed within polynomial time, thereby establishing 0-1 Integer Programming as a problem in NP.

2. **Reduction from SAT to 0-1 Integer Programming:**

Our aim here is to demonstrate that the SAT problem, known to be NP-complete, can be reformulated as a special case of 0-1 Integer Programming.
In SAT, Boolean variables and clauses exist, with each clause representing a disjunction of literals. The goal is to determine an assignment of Boolean values satisfying all clauses.
The reduction involves representing each Boolean variable X as a binary variable in 0 -1 Integer Programming:

X = 1, if true
  = 0, if false

To avoid explicit use of the complement, introduce two binary variables, Y_{i1} and Y_{i2}, for each original Boolean variable X_i , where:

$Y_{i1} + Y_{i2} = 1$

Clauses in SAT are then translated into inequalities in 0-1 Integer Programming, ensuring that the sum of relevant binary variables meets a specific threshold.
$\sum_{\text{literal l in clause C}} Y_l >= 1$

The solution to the 0-1 Integer Programming problem corresponds to a satisfying assignment for the SAT problem, and vice versa.
As SAT is NP-complete and we have successfully reduced it to 0-1 Integer Programming, we establish that 0-1 Integer Programming is NP-hard.
In conclusion, by demonstrating that 0-1 Integer Programming is both in NP and NP-hard through the reduction from SAT, we establish its status as an NP-complete problem.

Q2. (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.

Solution -

The clique problem is essentially the converse of the independent set problem. In a clique, all vertices are interconnected, forming a fully connected subgraph. On the other hand, an independent set is characterized by vertices that have no connections among them. To establish the NP-completeness of the clique problem through a reduction from the independent set problem, we can use a strategy that involves creating a complementary graph.

Let's assume the complement of a graph G as G'. The complement graph has the same set of vertices, but two vertices are adjacent in G' if and only if they are not adjacent in G.

Now, the reduction is as follows:

1. Given an instance (G, k) of the independent set problem, we construct the complement graph G'.
2. Set p = k.
3. The reduction claims that (G, k) has an independent set of size k if and only if (G', p) has a clique of size p.

Suppose we have an instance (G, p) of the clique problem. We can create the complement graph G' in polynomial time, as constructing G' involves examining each edge in G once. Next, we set the parameter for the independent set problem, k, equal to p. So, we need to check if G' have an independent set of size p.

If (G, k) has an independent set of size k, it implies that in G', there is a set of k vertices, each connected to every other vertex in the set. In other words, this set in G' forms a clique of size k. On the contrary, if (G', p) has an independent set of size p, it means that in G', there are p vertices, none of which are connected to each other. Considering this back to G, this set forms an independent set of size p since there are no edges between these vertices.

The construction of G' from G and the verification of the independent set in G' both take polynomial time, proving that the reduction process is efficient. Therefore, the clique problem is proven to be NP-complete through a polynomial-time reduction from the independent set problem.

Q3. (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.**

Solution –

Here, we have to demonstrate that the problem of determining if a directed graph has a directed Hamiltonian path (dHamPath) is NP-complete by reducing it from the known NP-complete problem of finding a directed Hamiltonian cycle (dHamCycle).

As we know, the dHamCycle involves determining whether a directed graph contains a directed cycle that visits each node exactly once. This problem is NP-complete.

Now the reduction is as follows,

1. We define a dHamCycle as it will help to check if a directed graph has a cycle that visits every node exactly once.
2. We define a dHamPath which involves determining whether a directed graph contains a directed path that visits every node exactly once.
3. We show the Reduction as
   Given an instance (G, k) of dHamCycle, where G is a directed graph and k is a positive integer, we construct an instance (G', s, t) of dHamPath.
   We construct G': Creating G' by introducing a new node x and connecting it to the starting node s and the target node t with edges (x, s) and (t, x).
   We adjust k and set k' to k + 1.

Here we can define the transformation as -
   If G has a directed Hamiltonian cycle, G' has a directed Hamiltonian path from s to t by traversing the cycle and using edges (x, s) and (t, x).
   If G' has a directed Hamiltonian path from s to t, removing edges (x, s) and (t, x) results in G having a directed Hamiltonian cycle.

4. Complexity Analysis
   The reduction is polynomial-time.
   Correctness can be given as: (G, k) is a "yes" instance of dHamCycle if and only if (G', s, t) is a "yes" instance of dHamPath.

Since dHamCycle is NP-complete, and we have successfully reduced it to dHamPath in polynomial time while preserving the answer, we can conclude that dHamPath is also NP-complete. This demonstrates the NP-completeness of the directed Hamiltonian path problem.

Q4. (20 points) Given a set of *n* items of values *V1, ...,Vn* and weights *W1,...,Wn*, 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.

Solution –

The Subset Sum problem involves deciding whether there exists a subset of a given set of integers S = {a1, a2, ..., an} whose sum equals a specified target T. Conversely, the 0-1 Knapsack problem requires selecting a subset of items, each characterized by a value Vi and weight Wi, to maximize the total value while adhering to a given capacity constraint W.

To establish the NP-hardness of the Knapsack problem, we can perform a reduction from the Subset Sum problem to the Knapsack problem. Let S = {a1, a2, ..., an} be an instance of the Subset Sum problem with target T. We construct an instance of the Knapsack problem as follows:

1. Create n items, each corresponding to an element in S: - Item i has value Vi = ai and weight Wi = ai for i = 1, 2, ..., n. Set the capacity of the knapsack to W = T.
2. The Knapsack algorithm is then applied to this constructed instance.

The key claim is that there exists a subset of S with a sum equal to T if and only if there exists a subset of items in the Knapsack instance with a total value of at least T.

Proof:

If there exists a subset of S with a sum equal to T, then selecting the corresponding items in the Knapsack instance (items with values equal to the selected elements in the subset) will yield a total value of at least T. Mathematically, if $S' \subseteq S$ and $\sum_{ai \in S'} ai = T$, then $\sum_{i=1}^{n} Vi \cdot xi >= T$, where xi is a binary variable indicating whether item i is selected.

If there exists a subset of items in the Knapsack instance with a total value of at least T, then selecting the corresponding elements in the original set S will give a subset with a sum equal to T. Mathematically, if { i | xi = 1 } is the set of selected items, then $\sum_{i \in \{ i \mid xi = 1\}} ai = \sum_{i=1}^{n} Wi \cdot xi = T$

Therefore, this reduction establishes the equivalence between the existence of a subset sum in S and a valuable subset in the Knapsack instance, demonstrating the NP-hardness of the Knapsack problem.

Q5. (20 points) Assume that there are *n* tasks with integer processing times *t1.. tn* 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.

Solution –

To show that finding out if the processing time can be equally divided in the scheduling problem is NP-complete, we need to perform a reduction from the Subset Sum problem, which is known to be NP-complete.

The Subset Sum Problem can be defined as - Given a set of integers S and a target integer T, does there exist a subset of S such that the sum of its elements is equal to T?

The Reduction can be shown as follows -
Let's construct an instance of the scheduling problem with two machines and tasks with integer processing times, such that the total processing time can be equally divided if and only if there exists a subset of S whose sum is equal to T.

1. Instance Construction: We set S for Subset Sum which will be represented as the set of processing times for the tasks: S = { t1, t2, ..., tn }. The target sum T will be half of the total processing time of all tasks: T = ($\sum^{n}_{i=2}$ ti) / 2

2. Scheduling Problem: As we have two machines, Machine 1 and Machine 2, we assign each task's processing time to a corresponding task on the machines. Let M1 and M2 represent the processing times on Machine 1 and Machine 2, respectively.

M1 = { t1, t2, ..., tn }
M2 = { 0,0, ..., 0 }

Our objective, here, is to minimize the total processing time on both machines.

3. Reduction Claim: There exists a subset of S whose sum is equal to T if and only if there exists a scheduling that divides the total processing time equally between the two machines.

Proof:
If there exists a subset of S whose sum is equal to T:
- This means there is a set of tasks whose processing times sum up to T.
- In the scheduling problem, this corresponds to a subset of tasks that can be assigned to Machine 1, while the remaining tasks are assigned to Machine 2.
- As a result, the total processing time on both machines is equal, and the scheduling problem is solvable.

If there exists a scheduling that divides the total processing time equally:
- This means there is a way to assign tasks to the machines such that the total processing time on each machine is the same.
- In the scheduling problem, this corresponds to a subset of tasks whose processing times sum up to T.
- Therefore, the Subset Sum problem is solvable.

This completes the reduction. Since the Subset Sum problem is known to be NP-complete, this reduction establishes that the scheduling problem is also NP-complete.