CS 170 Homework 11

Due **Monday 4/15/2024, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Some Sums

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $S \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in S} a_i = \sum_{j \in ([n] \setminus S)} a_j$. In other words, determine whether there is a way to partition A into two disjoint subsets such that the sum of the elements in each subset equal.

2 **Subset Sum**: Given some integer x, determine whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = x$. In other words, determine whether there is a subset of A such that the sum of its elements is x.

3 **Knapsack**: Given some set of items each with weight w_i and value v_i, and fixed numbers W and V, determine whether there is some subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$.

For each of the following clearly describe your reduction and justify its correctness.

(a) Find a linear time reduction from Subset Sum to Partition.

(b) Find a linear time reduction from Subset Sum to Knapsack.

Solution:

(a) Suppose we are given some A with target sum x. Let s be the sum of all elements in A. If $s - 2x \geq 0$, generate a new set $A' = A \cup \{s - 2x\}$. If A' can be partitioned, then there is a subset of A that sums to x.

We know that the two sets in our partition must each sum to $s - x$ since the sum of all elements will be $2s - 2x$. One of these sets, must contain the element $s - 2x$. Thus the remaining elements in this set sum to x.

If $s - 2x \leq 0$, generate a new set $A' = A \cup \{2x - s\}$. If A' can be partitioned, then there is a subset of A that sums to x.

We know that the two sets in our partition must each sum to x since the sum of all elements will be $2x$. The set that does not contain $\{2x - s\}$ will be our solution to subset sum.

(b) Suppose we are given some set A with target sum x. For each element i of the set, create an item with weight i and value k. Let $V = x$ and $W = x$. We know Knapsack will determine if there is a combination of items with sum of weights $\leq x$ and values

$\geq x$. Because the weights and values are the same, we know (Sum of chosen weights) = (Sum of chosen values) $= x$. And since each weight/value pair is exactly the value of one of the original elements of A, we know that there will be a solution to our Knapsack problem iff there is one for our subset sum problem.

3 k-XOR

In the k-XOR problem, we are given n boolean variables $x_1, x_2, \ldots, x_n$, a list of m clauses each of which is the XOR of exactly k distinct variables (that is, the clause is true if and only if an odd number of the k variables in the clause are true), and an integer c. Our goal is to decide if there is some assignment of variables that satisfies at least c clauses.

(a) In the Max-Cut problem, we are given an undirected unweighted graph $G = (V, E)$ and integer α and want to find a cut $S \subseteq V$ such that at least α edges cross this cut (i.e. have exactly one endpoint in S). Give and argue correctness of a reduction from Max-Cut to 2-XOR.

Hint: every clause in 2-XOR is equivalent to an edge in Max-Cut.

(b) Give and argue correctness of a reduction from 3-XOR to 4-XOR.

Solution:

(a) We create a variable for each vertex i, and a clause x_i XOR x_j for each edge (i, j). We choose $\alpha = c$.

For any assignment, consider the cut such that S contains all vertices i for which x_i is true in this assignment. For each edge (i, j) crossing this cut, its corresponding clause is true because exactly one of x_i, x_j is true. For each edge not crossing this cut, its corresponding clause is false because either both x_i, x_j are true or neither is true. This proves correctness of the reduction.

(b) The reduction is to add a new variable z and add z to every clause in the Max 3-XOR instance and choose the same value of c to get a Max 4-XOR instance.

Given an assignment that satisfies c clauses in the Max 3-XOR instance, the same assignment plus z set to false satisfies c clauses in the Max 4-XOR instance. Given an assignment that satisfies c clauses in the Max 4-XOR instance, if z is false, then the same assignment minus z satisfies c clauses in the Max 3-XOR instance. Otherwise, z is true, and the same assignment minus z and negating all other variables satisfies c clauses in the Max 3-XOR instance.

To see this, consider any clause that was true in Max 4-XOR. Deleting z takes it from having an odd number of true variables to an even number. Then, since 3 is odd, negating all variables brings it back to having an odd number of true variables. Similarly, any clause that was false in the Max 4-XOR instance has an even number of true variables. Deleting z causes it to have an odd number of true variables, and then negating all other variables brings it back to even.

4 Survivable Network Design

Survivable Network Design is the following problem:

> We are given two $n \times n$ matrices: a cost matrix d_{ij} and a connectivity requirement matrix r_{ij}, both of which are symmetric. We are also given a budget b. Our goal is to find an undirected graph $G = (\{1, ..., n\}, E)$ such that the total cost of all edges (i.e. $\sum_{(i,j)\in E} d_{ij}$) is at most b and there are exactly r_{ij} edge-disjoint paths between any two distinct vertices i and j; if no such G exists, output "None". (A set of paths is edge-disjoint if no edge appears in more than one of them)

Show that Survivable Network Design is NP-Complete.

Hint: Reduce from a NP-Hard problem in Section 8 of the textbook.

Solution: We first show that Survivable Network Design is in NP. Given a solution to Survivable Network Design (i.e. a graph G), we can check the budget constraint in linear time easily. To check the connectivity requirements, we can check if the max flow between every pair i, j using unit capacities on the edges is at least r_{ij}. Any max-flow of value F can be decomposed into F edge-disjoint i-j paths, since all capacities are unit.

Now, we show that Survivable Network Design is in NP-Hard. To do this, we use the following reduction from Rudrata Cycle to Survivable Network Design:

> Given input $G = (V, E)$ to Rudrata Cycle, construct the corresponding Survivable Network Design instance as follows: set $d_{ij} = 1 \ \forall (i, j) \in E$ and $d_{ij} > n$ otherwise; set $r_{ij} = 2 \ \forall (i, j)$; and set $b = n$ (recall $|V| = n$).

($\Rightarrow$) Clearly any Rudrata Cycle is a solution to this Survivable Network Design instance. So we just need to show that any solution in this instance is a Rudrata Cycle.

($\Leftarrow$) Any solution uses edges such that the sum of the weights of all the edges is at most n, i.e. uses at most n edges since all usable edges have weight 1. In order to have two edge-disjoint paths between any two distinct vertices, every vertex must have degree at least 2. But since we can only use n edges, the total degree of all vertices can't be more than $2n$. So every vertex has degree exactly 2. This means that the edges must form a set of disjoint cycles. But every pair of vertices needs to be connected by the edges, so this set of disjoint cycles must actually be a single cycle visiting every vertex, i.e. a Rudrata Cycle.

5 Orthogonal Vectors

In the 3-SAT problem, we have n variables and m clauses, where each clause is the OR of (at most) three of these variables or their negations. The goal of the problem is to find an assignment of variables that satisfies all the clauses, or correctly declare that none exists.

In the orthogonal vectors problem, we have two sets of vectors A, B. All vectors are in $\{0,1\}^m$, and $|A| = |B| = n$. The goal of the problem is to find two vectors $a \in A, b \in B$ whose dot product is 0, or correctly declare that none exists. The brute-force solution to this problem takes $O(n^2 m)$ time: compute all $|A||B| = n^2$ dot products between two vectors in A, B, and each dot product takes $O(m)$ time.

Show that if there is a $O(n^c m)$-time algorithm for the orthogonal vectors problem for some $c \in [1, 2)$, then there is a $O(2^{cn/2} m)$-time algorithm for the 3-SAT problem. For simplicity, you may assume in 3-SAT that the number of variables must be even.

Hint: Try splitting the variables in the 3-SAT problem into two groups.

Solution: We use an $O(2^{n/2} m)$-time reduction from 3-SAT to orthogonal vectors. We split the variables into two groups of size $n/2$, V_1 and V_2. For each group, we enumerate all $2^{n/2}$ possible assignments of these variables. For each assignment x of the variables in V_1, let v_x be the vector where the ith entry is 0 if the ith clause is satisfied by one of the variables in this assignment, and 1 otherwise. We ignore the variables in the clause that are in V_2. For example, if clause i only contains variables in V_2, then $v_x(i) = 0$ for all x. Let A be the $2^{n/2}$ vectors produced this way.

We construct B containing $2^{n/2}$ vectors in a similar manner, except using V_2 instead of V_1.

We claim that the 3-SAT instance is satisfiable if and only if there is an orthogonal vector pair in $A \times B$. Given this claim, we can solve 3-SAT by making the orthogonal vectors instance in $O(2^{n/2} m)$ time, and then solving the instance in $O((2^{n/2})^c m) = O(2^{cn/2} m)$ time.

Suppose there is a satisfying assignment to 3-SAT. Let x_1 be the assignment of variables in V_1, and x_2 be the assignment of variables in V_2. Let v_1, v_2 be the vectors in A, B corresponding to x_1, x_2. Since every clause is satisfied, one of $v_1(i)$ and $v_2(i)$ must be 0 for every i, and so $v_1 \cdot v_2 = 0$. So there is also a pair of orthogonal vectors in the orthogonal vectors instance.

Suppose there is a pair of orthogonal vectors v_1, v_2 in the orthogonal vectors instance. Then for every i, either $v_1(i)$ or $v_2(i)$ is 0. In turn, for the corresponding assignment of variables in V_1, V_2, the combination of these assignments must satisfy every clause. Hence, the combination of these assignments is a satisfying assignment for 3-SAT.

Comment: It is widely believed that SAT has no $O(2^{.999n} m)$-time algorithm - this is called the Strong Exponential Time Hypothesis (SETH). So it is also widely believed that orthogonal vectors has no $O(n^{1.99} m)$-time algorithm, since otherwise SETH would be violated. It turns out that we can reduce orthogonal vectors to string problems such as edit distance and longest common subsequence, and so if we belive SETH then we also believe those problems also don't have $O(n^{1.99})$-time algorithms. The field of research studying reductions between problems with polynomial-time algorithms such as these is known as fine-grained complexity, and orthogonal vectors is one of the central problems in this field.