CS 600 A - Advanced Algorithms – Homework 9

Name: Deepali Nagwade | 20013393

Question 17.8.8: Given the CNF formula B = (x) . (x2 + x3 + x5 +) . (x1 + 24) . (X3 + X5), show the reduction of B into an equivalent input for the 3SAT problem.

Answer: To convert a clause containing less than 3 literals, we will add free variables so that the clause have 3 literals.

And the clause containing more than 3 literals, we will break the clause in two or more parts such that each clause should have 3 literals while adding one or more free variables and satisfiability of original clause and new clauses should be equivalent.

And the clause containing less than 3 literals, we will add free variable to make it clause of 3 literals and creating additional clause such that satisfiability of product of clause depend only on satisfiability of original clause.

So in the given CNF formula B, the first clause is (x_1) having one literals, so we add two free variables $y_1 \text{ and } y_2$ to make it $(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \bar{y_2}) \wedge (x_1 \vee \bar{y_1} \vee y_2) \wedge (x_1 \vee \bar{y_1} \vee \bar{y_2})$. So if (x_1) is satisfiable then $(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \bar{y_2}) \wedge (x_1 \vee \bar{y_1} \vee y_2) \wedge (x_1 \vee \bar{y_1} \vee \bar{y_2})$ is satisfiable and vice-versa.

$(\bar{x_2} + x_3 + x_5 + \bar{x_6})$ will be break into $(\bar{x_2} \vee x_3 \vee y_3) \wedge (\bar{y_3} \vee x_5 \vee \bar{x_6})$.

$(x_1 + x_4)$ will be transformed to $(x_1 \vee x_4 \vee y_4) \wedge (x_1 \vee x_4 \vee \bar{y_4})$.

$(x_3 + \bar{x_5})$ will be transformed to $(x_3 \vee \bar{x_5} \vee y_5) \wedge (x_3 \vee \bar{x_5} \vee \bar{y_5})$.

Hence the 3SAT formula equivalent to above is obtaining by combining all clause using AND operator :

$$(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \bar{y_2}) \wedge (x_1 \vee \bar{y_1} \vee y_2) \wedge (x_1 \vee \bar{y_1} \vee \bar{y_2}) \wedge (\bar{x_2} \vee x_3 \vee y_3) \wedge$$
$$(\bar{y_3} \vee x_5 \vee \bar{x_6}) \wedge (x_1 \vee x_4 \vee y_4) \wedge (x_1 \vee x_4 \vee \bar{y_4}) \wedge (x_3 \vee \bar{x_5} \vee y_5) \wedge (x_3 \vee \bar{x_5} \vee \bar{y_5})$$

Question 17.8.18: Professor Amongus has just designed an algorithm that can take as input any graph **G** with **n** vertices and a number **k** and determine in **O**(**nk**) time whether **G** contains a clique of size k. Does Professor Amongus deserve the Turing Award for having just shown that *P=NP*? Why or why not?

Answer: The claim that Professor Amongus has designed an algorithm that can determine whether a graph contains a clique of size k in O(n^k) time is a significant one. However, it is important to note that this claim alone does not prove that P=NP or that Professor Amongus deserves the Turing Award. Let me explain further.

The P versus NP problem is one of the most significant unsolved problems in computer science. It asks whether every problem for which a solution can be verified in polynomial time can also be solved in polynomial time. In simpler terms, it investigates whether P (problems that can be efficiently solved) is equal to NP (problems that can be efficiently verified).

Explanation:
The statement "P=NP" refers to the question of whether the complexity class P is equal to the complexity class NP. To understand this statement, let's break down what P and NP represent:

- P (Polynomial Time): P represents the class of decision problems that can be solved by a deterministic Turing machine in polynomial time. In other words, these are problems for which an algorithm exists that can provide a solution in a reasonable amount of time, typically represented as a polynomial function of the problem's input size.

- NP (Nondeterministic Polynomial Time): NP represents the class of decision problems for which a solution can be verified in polynomial time. In other words, if someone claims to have a solution to an NP problem, it can be checked efficiently to determine whether it is correct or not. However, finding the solution itself may not be efficient.

While Professor Amongus claims to have designed an algorithm with a time complexity of O(n^k) for determining cliques, it is crucial to subject the algorithm to rigorous analysis and peer review. The time complexity itself does not necessarily provide evidence for P=NP or guarantee the correctness of the claim. It is possible that the algorithm has limitations, assumptions, or conditions that affect its applicability or efficiency in practice.

For Professor Amongus to be considered for the Turing Award, their algorithm would need to undergo thorough scrutiny and validation by the scientific community. The Turing Award is typically granted for significant and groundbreaking contributions to the field of computer science, and extraordinary claims require extraordinary evidence. It would require a consensus among experts and substantial peer-reviewed research before such a claim could be deemed award-worthy.

In summary, the claim made by Professor Amongus should be subjected to rigorous scrutiny and validation by the scientific community. While the algorithm's time complexity is noteworthy, it alone does not prove P=NP or guarantee the Turing Award.

However, based solely on the information given, it would not be sufficient to prove P=NP and therefore may not be considered for the Turing Award on its own.

Question 17.8.28 Explain Hyper-Community to be the problem that takes a collection of n web pages and an integer k, and determines if there are k web pages that all contain hyperlinks to each other. Show that this problem is NP-complete.

Answer: To show that the HYPER-COMMUNITY problem is NP-complete, we need to demonstrate two things:

- HYPER-COMMUNITY is in NP.
- HYPER-COMMUNITY is NP-hard, i.e., every problem in NP can be reduced to HYPER-COMMUNITY in polynomial time.

Let's go through these two steps:

1. **HYPER-COMMUNITY is in NP:**
To show that HYPER-COMMUNITY is in NP, we need to demonstrate that, given a proposed solution (a set of web pages forming a hyper-community), we can verify its correctness in polynomial time.

A possible certificate for a solution is a set of web pages, and to verify it, we can check that each web page in the set has hyperlinks to all other web pages in the set. This verification can be done in polynomial time, as it involves checking links between a polynomial number of pairs of web pages.

2. **HYPER-COMMUNITY is NP-hard:**
To show that HYPER-COMMUNITY is NP-hard, we can reduce the well-known NP-complete problem CLIQUE to HYPER-COMMUNITY in polynomial time.

The CLIQUE problem is defined as follows: Given an undirected graph G and an integer k, determine whether there exists a clique (a subset of vertices where every pair of distinct vertices is connected by an edge) of size k in G.

Let's construct a polynomial-time reduction from CLIQUE to HYPER-COMMUNITY:

Given an instance of CLIQUE with graph G and integer k, we create an instance for HYPER-COMMUNITY as follows:

1. Each vertex in G corresponds to a web page in HYPER-COMMUNITY.
2. For every edge in G, create a hyperlink between the corresponding web pages in HYPER-COMMUNITY.

Now, if there exists a clique of size k in G, then the corresponding web pages in HYPER-COMMUNITY form a hyper-community. Conversely, if there is a hyper-community in HYPER-COMMUNITY, it corresponds to a clique of size at least k in G.

This reduction can be done in polynomial time, and therefore, we have shown that HYPER-COMMUNITY is NP-hard.

Since HYPER-COMMUNITY is both in NP and NP-hard, it is NP-complete.

Question 17.8.35 Imagine that the annual university job fair is scheduled for next month and it is your job to book companies to host booths in the large Truman Auditorium during the fair. Unfortunately, at last year's job fair, a fight broke out between some people from competing companies, so the university president, Dr. Noah Drama, has issued a rule that prohibits any pair of competing companies from both being invited to this year's event. In addition, he has shown you a website that lists the competitors for every company that might be invited to this year's job fair and he has asked you to invite the maximum number of noncompeting companies as possible. Show that the decision version of the problem Dr. Drama has asked you to solve is NP-complete.

Answer: To show that the decision version of the problem Dr. Drama has asked you to solve is NP-complete, we need to prove two things: the problem is in NP, and the problem is NP-hard. We'll call this problem the Non-Competing Company Invitation Problem (NCCIP).

1. **The problem is in NP:**
We can demonstrate that NCCIP is in NP by showing that, given a certificate (a potential solution), we can verify whether it is a correct solution in polynomial time. In this case, the certificate would be a list of companies to invite to the job fair.

Explanation**:**
Given a list of invited companies, we can check in polynomial time whether any pair of companies in the list are competitors (by looking up the competitors for each company on the provided website). If there are no competitors in the list and the number of invited companies is equal to or greater than the target number, then the certificate is a valid solution.

2**. The problem is NP-hard:**

To show that NCCIP is NP-hard, we need to reduce a known NP-complete problem to NCCIP in polynomial time. We will use the Maximum Independent Set (MIS) problem for this reduction, which is a well-known NP-complete problem.

In the Maximum Independent Set problem, we are given an undirected graph G = (V, E) and an integer k. The problem asks whether there is an independent set of size at least k in G, i.e., a set of vertices such that no two vertices in the set are adjacent.

Reduction: Given an instance of the MIS problem with graph G = (V, E) and integer k, we can construct an instance of the NCCIP as follows:

- Each vertex in V represents a company.
- Each edge in E represents a pair of competing companies.
- The target number of noncompeting companies to invite is k.

This reduction can be done in polynomial time since we only need to create a mapping from vertices to companies and edges to competitor relationships.
Now, let's prove the reduction is correct.

- If there is an independent set of size at least k in G, then there is a set of k noncompeting companies that can be invited. In this case, no two companies in the independent set are competitors since there is no edge between them in the graph.
- If there is a set of k noncompeting companies that can be invited, then there is an independent set of size at least k in G. No two invited companies are competitors, so there is no edge between them in the graph.

Since we have shown that NCCIP is in NP and NP-hard, we can conclude that the decision version of the Non-Competing Company Invitation Problem is NP-complete.

Question 18.6.19 In the ***Euclidean*** traveling salesperson problem, cities are points in the plane and the distance between two cities is the Euclidean distance between the points for these cities, that is, the length of the straight line joining these points. Show that an optimal solution to the Euclidean TSP is a simple polygon, that is, a connected sequence of line segments such that no two ever cross.

Answer: Cities are points in the plane in the Euclidean travelling salesperson problem, and the distance between two cities is the Euclidean distance between the points for these cities, that is, the length of the straight line connecting these points. Show that a simple polygon is an optimal solution to the Euclidean TSP, that is, a connected sequence of line segments that never cross.

This is because the straight line is the shortest path between any two points, therefore a sequence of straight lines cannot cross.

Explanation:

- One of the most well-known problems in the fields of computer technology and mathematics is called the Euclidean TSP. The objective of this issue is to determine, given a collection of points in the plane, the shortest route that goes from one point to every other point in the set exactly once and then back to the initial point. This problem is known to be NP-hard, which means that there is no algorithm that is known to exist that is capable of solving it in a time that is polynomial. On the other hand, there are a number of heuristics and approximation algorithms that are capable of finding solutions in practise that are acceptable or better.

- The nearest neighbour technique is the most straightforward solution to the Euclidean travelling salesman problem. This algorithm begins at the first point in the set that has been provided, and it continues to add the point that is geographically closest to the path that is being followed until all points have been added. This algorithm is guaranteed to discover a path that is no more than twice as long as the best path, but in practise, it typically produces paths that are much shorter than the optimal path.

- The Christofides algorithm is another another well-known heuristic that can be used for the Euclidean TSP. This approach begins by computing a minimum spanning tree for the set of points that it has been provided with. After that, it locates the shortest route that still travels to each of the locations in the tree. Last but not least, it includes any points that were missing from the tree. This algorithm is guaranteed to find a path that is no longer than 1.5 times as long as the path that is deemed to be optimal.

- In addition, there are a few different approximation procedures that can be used for the Euclidean Traveling Salesman Problem. These methods will always find a road that is shorter than the optimal path length, but they do not ensure that the path they find will be the shortest path possible. The 2-opt algorithm is widely considered to be one of the most successful approximation techniques. This algorithm begins with a path that has already been determined, and it then continually swaps two edges inside the path if doing so would result in a shorter path length. This method is done multiple times until there are no more possible improvements. It is a property of the 2-opt algorithm that it will always find a path that is no longer than two times as long as the optimal path.

  Heuristics and approximation algorithms are frequently used in conjunction with one another in order to solve the Euclidean TSP in practise. These algorithms are able to develop solutions to the problem that are acceptable for the most part within a reasonable length of time.

Question 18.6.26 Suppose you work for a major package shipping company, FedUP, as in the previous exercise, but suppose there is a new law that requires every truck to carry no more than M pounds, even if it has room for more boxes. Now the optimization problem is to use the fewest number of trucks possible to carry the n boxes across the country such that each truck is carrying at most M pounds. Describe a simple greedy algorithm for assigning boxes to trucks and show that your algorithm uses a number of trucks that is within a factor of 2 of the optimal number of trucks. You may assume that no box weighs more than M pounds.

Answer: Here's a simple greedy algorithm for assigning boxes to trucks, ensuring no truck exceeds the weight limit M:

- Initialize an empty list of trucks, each with a weight capacity of M pounds.
- Process the boxes in the order they arrive. For each box:

  a) Check if the current truck has enough remaining capacity to accommodate the box without exceeding the weight limit.

     i. If yes, add the box to the current truck.
     ii. If no, proceed to step b.

  b) If the current truck is full or no more boxes can fit without exceeding the weight limit, create a new truck and add the box to it.

- Repeat step 2 until all boxes have been assigned to trucks.

To show that this algorithm uses a number of trucks within a factor of 2 of the optimal number of trucks, consider an optimal solution that uses 'k' trucks. In this optimal solution, each truck must carry at least M/2 pounds of boxes, as otherwise, a box could be moved to another truck, reducing the total number of trucks required.

Our greedy algorithm may use up to twice as many trucks as the optimal solution. This occurs when the optimal solution packs boxes tightly, ensuring that each truck carries close to the weight limit M. In this scenario, our greedy algorithm may create new trucks prematurely, leading to a total of up to 2k trucks.

However, our greedy algorithm is guaranteed to use at most twice as many trucks as the optimal solution. This demonstrates its efficiency and effectiveness in approximating the optimal solution.

Example : Suppose we have the following boxes and trucks:

Boxes: Box 1: 10 pounds, Box 2: 15 pounds, Box 3: 5 pounds, Box 4: 10 pounds, Box 5: 5 pounds

Trucks: Truck 1: Weight capacity of 20 pounds, Truck 2: Weight capacity of 20 pounds

Greedy Algorithm:

Process Box 1: Box 1 weighs 10 pounds, and Truck 1 has a remaining capacity of 20 pounds. Therefore, add Box 1 to Truck 1.

Process Box 2: Box 2 weighs 15 pounds, and Truck 1 only has a remaining capacity of 10 pounds. Therefore, create a new truck (Truck 2) and add Box 2 to it.

Process Box 3: Box 3 weighs 5 pounds, and Truck 2 has a remaining capacity of 5 pounds. Therefore, add Box 3 to Truck 2.

Process Box 4: Box 4 weighs 10 pounds, and Truck 2 is full. Therefore, create a new truck (Truck 3) and add Box 4 to it.

Process Box 5: Box 5 weighs 5 pounds, and Truck 3 has a remaining capacity of 15 pounds. Therefore, add Box 5 to Truck 3.

Final Assignment:

Truck 1: Box 1

Truck 2: Box 2, Box 3

Truck 3: Box 4, Box 5

In this example, the greedy algorithm used 3 trucks, while the optimal solution would use 2 trucks. However, the greedy algorithm is still within a factor of 2 of the optimal solution.