Homework 3

Ameyassh Nagarajan

April 2024

1 Question 1

Prove that the following problem is coNP-complete:

$$\text{EQV} = \{(\phi_1, \phi_2) \mid \phi_1 \text{and} \phi_2 \text{ are logically equivalent boolean formulas}\}$$

EQV is in coNP

The problem EQV is defined as determining whether two Boolean formulas, ϕ_1 and ϕ_2, are logically equivalent, meaning they produce the same results for all variable assignments. A problem is in coNP if the complement of the problem can be verified in polynomial time.

Complement of EQV

The complement of EQV involves pairs of Boolean formulas (ϕ_1, ϕ_2) that are not equivalent. To verify non-equivalence, one needs to find an assignment of the variables where the evaluations of ϕ_1 and ϕ_2 differ.

Verification Process

1. Guess an assignment of the variables.
2. Evaluate both ϕ_1 and ϕ_2 under this assignment.
3. Check if the outputs differ.

This process can be executed in polynomial time, confirming that the non-equivalence can be verified efficiently, thereby placing EQV in coNP.

Definitions

Tautology (TAUT)

The Tautology problem involves determining whether a Boolean formula, denoted ϕ, evaluates to true for every possible assignment of its variables.

Equivalence (EQV)

The Equivalence problem requires establishing whether two Boolean formulas, ϕ_1 and ϕ_2, are logically equivalent. Two formulas are considered equivalent if they yield the same truth value under every possible assignment of their variables. Moreover, for the purpose of this proof, both formulas must share the exact same set of variables.

Reduction from TAUT to EQV

Idea for the reduction

The main idea for the reduction is to convert to a TAUT to an EQV problem. To show that, solving EQV is atleast as hard as TAUT, which is a well known coNP-complete problem.

Constructing the Equivalence Test

To perform this reduction, we employ the following steps:

- Let $\phi_1 = \phi$, where ϕ is the formula to be tested for tautology.
- Construct ϕ_2 to be explicitly tautological and include all variables present in ϕ_1. This ensures that both formulas share the same variable set. For simplicity and to maintain generality, let:

$$\phi_2 = \bigwedge_{i=1}^{n} (x_i \vee \neg x_i),$$

  where x_i are the variables in ϕ. This construction guarantees that ϕ_2 is tautologically true.

Logical Implication of the Reduction

By constructing ϕ_1 and ϕ_2 as above, the equivalence $\phi_1 \equiv \phi_2$ serves as a direct test for the tautological nature of ϕ_1. Specifically:

- If $\phi_1 \equiv \phi_2$, then ϕ_1 must evaluate to true under all variable assignments, just like ϕ_2, confirming that ϕ_1 is a tautology.
- Conversely, if ϕ_1 is not a tautology, there exists at least one assignment under which ϕ_1 evaluates to false, thereby breaking its equivalence to the always true ϕ_2.

Polynomial Time Reduction

The reduction from the Tautology problem (TAUT) to the Equivalence problem (EQV) takes polynomial time, proving that it is a valid Karp reduction. The process involves:

- **Copying the TAUT Formula:** Directly replicating the tautology formula, ϕ, as ϕ_1. This step is linear with respect to the length of ϕ.
- **Constructing ϕ_2:** Formulating ϕ_2 as $(x_1 \vee \neg x_1) \wedge \ldots \wedge (x_n \vee \neg x_n)$, which scales linearly with the number of variables, n, in ϕ_1.

Both steps ensure the reduction is completed in polynomial time.

Conclusion

This reduction shows that EQV is at least as hard as TAUT and that solving EQV will yield a result for TAUT this also goes on to show that EQV is coNP-Hard. And since EQV is in coNP and it is coNP-Hard, EQV is coNP-Complete.

2 Question 2

Problem 2: Show that the following problem is NP-complete: Given a set of linear inequalities over a set of variables $x_1, \ldots, x_n$, determine whether there is an assignment of integers to the x_i's that satisfies all inequalities.

Example:

$$\begin{aligned} x_1 + 3x_2 - x_5 &\geq 3 \\ 2x_2 - x_4 &\leq x_3 \\ x_1 + x_2 + x_3 + x_4 + x_5 &\geq 0 \end{aligned}$$

This system of inequalities is satisfied by $x_1 = 10$, $x_2 = 3$, $x_3 = x_4 = -3$, $x_5 = 0$ (among many others).

Defining the Problem

Let us begin by calling this language as **INEQ**, it is defined such that:

$$\text{INEQ} = \left\{ (A, \vec{b}, \vec{c}) \mid \text{There exists } \vec{x} \in \mathbb{Z}^n \text{ such that } A\vec{x} \leq \vec{b} \text{ and } A\vec{x} \geq \vec{c} \right\}$$

where A is a matrix of coefficients, $\vec{b}$ and $\vec{c}$ are vectors of bounds, and $\vec{x}$ is a vector of integer variables.

INEQ is in NP

To classify a problem as belonging to NP, we need to establish that any proposed solution to the problem can be verified efficiently—specifically, in polynomial time relative to the size of the input. This does not necessarily imply that finding a solution is easy, but that checking a given solution is computationally manageable.

we focus on how to verify a given solution effectively.

Verification Steps for INEQ:

- **Matrix-Vector Multiplication:** Start by computing the product $A\vec{x}$, where A is a matrix and $\vec{x}$ is a vector of integers. This calculation involves a series of multiplications and additions based on the dimensions of A and $\vec{x}$, which is typically a polynomial-time operation.
- **Comparison Against Bounds:** Once $A\vec{x}$ is computed, each component of the resulting vector must be compared against the corresponding components of vectors $\vec{b}$ and $\vec{c}$. This step checks if each inequality $A\vec{x} \leq \vec{b}$ and $A\vec{x} \geq \vec{c}$ is satisfied.

These operations, though potentially numerous, are straightforward and scale polynomially with the input size. This is why verifying a candidate solution for the INEQ problem is efficient and confirms that INEQ is appropriately classified as an NP problem.

Reduction

Idea for the reduction

The main idea for the reduction is to convert to a **CNF** to an inequality. To show that, solving **INEQ** is atleast as hard as **SAT**, which is a well known NP-complete problem.

Reduction from SAT to INEQ

Given a CNF formula ψ consisting of n variables and m clauses, each clause can be written as a disjunction of literals. Each literal is either a variable x_i or its negation $\neg x_i$. Our goal is to translate each clause into a linear inequality.

Transformation Steps

1. **Representation of Literals:** Each variable x_i in the formula is represented as a binary variable in the inequality system, taking values in $\{0, 1\}$. The negation of a variable, $\neg x_i$, is represented as $1 - x_i$. This ensures that if $x_i = 1$, then $\neg x_i = 0$, and if $x_i = 0$, then $\neg x_i = 1$.

2. **Clause to Inequality Conversion:** Consider a generic clause $C_j = (l_{j1} \vee l_{j2} \vee \ldots \vee l_{jk})$. We convert this clause into an inequality as follows:

   If $l_{jt} = x_i,$ it contributes x_i to the inequality.
   If $l_{jt} = \neg x_i,$ it contributes $1 - x_i$ to the inequality.
   Additionally, for every literal l_{jt},
   the corresponding variable x_i is bounded to take values only in $\{0, 1\}$,
   which is enforced by the inequalities $0 \leq x_i$ and $x_i \leq 1$.

   Thus, the inequality for the clause C_j is:

   $$\sum_{t=1}^{k} \text{value}(l_{jt}) \geq 1,$$

   where value(l_{jt}) is x_i if l_{jt} is x_i, and $1 - x_i$ if l_{jt} is $\neg x_i$.

3. **Polynomial Time Reduction:** Constructing each inequality involves a number of operations linear in the number of literals in each clause. Since each literal contributes one term to the inequality (either x_i or $1 - x_i$), the transformation of each clause into an inequality can be done in polynomial time relative to the size of the clause. Thus, the entire transformation of the CNF formula into a system of inequalities is polynomial in the size of the input formula ψ.

Conclusion

This reduction demonstrates that solving INEQ (checking whether there exists a solution to a system of linear inequalities) is at least as computationally hard as solving SAT, since every instance of SAT can be translated into an instance of INEQ in polynomial time. Therefore, INEQ is NP-hard. Additionally since, INEQ is in NP and NP-Hard, INEQ is NP-Complete.

3 Question 3

Consider the following puzzle game. The game board is a grid of $n \times m$ squares. Some of the squares initially contain a token, which can be either red or blue. To solve the puzzle, you must remove a subset of the tokens so that:

1. Every row has at least one token remaining.
2. No column has tokens of different colors remaining.

The objective is to prove that it is NP-complete to decide whether such a puzzle can be solved.

Examples

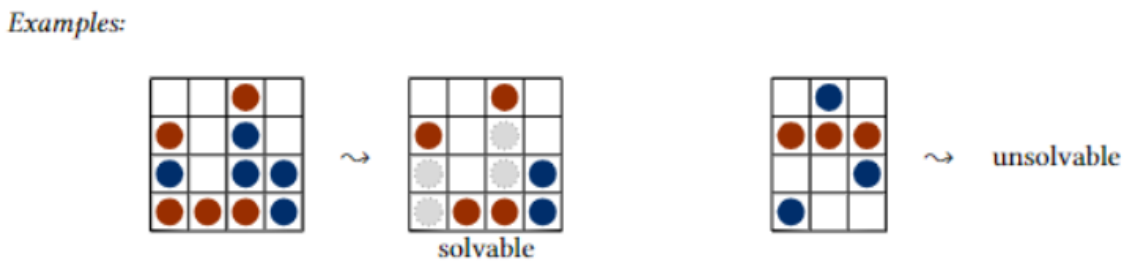

Figure 1: From left to right: A solvable configuration, a solvable configuration, and an unsolvable configuration

We consider a puzzle game where the board is a grid of $n \times m$ squares, with each square containing a token that is either red or blue. The objective is to determine whether there exists a subset of tokens that can be removed such that:

1. Every row has at least one token remaining.
2. No column contains tokens of different colors.

Proof of NP-Completeness

Problem Definition

Define the decision problem associated with the puzzle as follows:

PUZZLE-GAME = $\{(T, n, m) \mid \exists$ a valid subset of tokens T in an $n \times m$ grid satisfying the conditions$\}$

NP Membership

To show that PUZZLE-GAME is in NP, observe that any certificate or solution, which is a subset of tokens remaining on the board, can be verified in polynomial time. A verifier would need to check that:

- Each row contains at least one token.
- No column contains both a red and a blue token simultaneously.

These steps can be done in a polynomial amount of time.

NP-Hardness

To prove NP-hardness, we reduce the 3SAT problem (known to be NP-complete) to the PUZZLE-GAME problem. Consider an instance of 3SAT ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.

Reduction from 3SAT to a PUZZLE-GAME

To illustrate the reduction from a 3SAT problem to a satisfiability puzzle, consider the following construction:

Grid Construction:

- Construct a grid with one row for each variable and one column for each clause in the 3SAT instance. Each cell in this grid, denoted as $Cell[i, j]$, is capable of holding two tokens: one red and one blue.
- The red token in row i represents the positive literal x_i being part of clause C_j, whereas the blue token represents the negated literal $\overline{x_i}$ in the same clause.
- The overall time complexity of this will be $O(n \times m)$ where n is the number of variables or literals and m is the number of clauses. We get this as a result of constructing the $n \times m$ table from the instance of our 3SAT formula.

Token Manipulation Rules:

- Each token in $Cell[i, j]$ indicates the presence of a literal in clause C_j. Removing a blue token implies setting the variable x_i to `false` for satisfying C_j and to `true` otherwise, and removing a red token means setting $\overline{x_i}$ to `false` and `true` otherwise.
- This manipulation allows for a visual and interactive representation of variable assignments that satisfy the given 3SAT instance.

Ensuring Satisfiability:

- To ensure a satisfying assignment, each row must retain at least one token. This corresponds to consistently assigning truth values across all clauses in which the variable appears.

- The configuration must also ensure that no column contains tokens of both colors, representing a scenario where a variable and its negation are both set to true, which is logically inconsistent.

This reduction effectively transforms the logical structure of a 3SAT problem into an instance of PUZZLE-GAME of tokens, where the challenge is to satisfy all clauses by appropriate token removals, thereby mapping directly to the satisfaction of the corresponding 3SAT instance. And since this reduction can be done in an amount of time that is a polynomial function of the input size of ϕ we can say that 3SAT is karp reducible to PUZZLE-GAME and since PUZZLE-GAME is in NP and NP-Hard, we can say that PUZZLE-GAME is NP-Complete.

Conclusion

Since we can reduce 3SAT to PUZZLE-GAME in polynomial time and verify solutions in polynomial time, PUZZLE-GAME is NP-complete.