CS 3510: Design & Analysis of Algorithms 10/27/2024

Problem Set 7: NP-completeness I

Prof. Abrahim Ladha Due: 11/4/2024 11:59pm

- **Please TYPE your solutions using Latex or any other software. Handwritten solutions *won't* be accepted.**

Steps to write a reduction

To show that a problem is NP-complete, you need to show that the problem is both in NP and NP-hard. In a polynomial-time reduction, we have a *Problem B*, and we want to show that it is NP-complete. These are the steps:

- Demonstrate that *problem B* is in the class NP. This is a description of a procedure that verifies a candidate's solution. It needs to address the runtime. You should give a polytime verifier which takes as input a candidate problem, and a witness, and outputs true or false if the witness is a solution.
- Demonstrate that *problem B* is at least as hard as a problem previously proved to be NP-complete. Choose some NP-complete A and prove $A \leq_p B$. You need to show that *problem B* is NP-hard. This is done via polytime reduction from a known NP-complete *problem A* to the unknown textitproblem B ($A \to B$) as follows:
  1. Choose A. You will have many NP-complete problems to pick from later on, and it is best to pick a similar one.
  2. Give your reduction f and argue that it runs in polynomial time.
  3. Prove that $x \in A \implies f(x) \in B$
  4. Prove that $x \notin A \implies f(x) \notin B$
  5. This is sufficient to show $x \in A \iff f(x) \in B$. Since A was NP- complete, and $A \leq_p B$, we can conclude that B is NP-hard. Note: You don't have to provide a formal proof. You can briefly explain in words both implications and why they hold.

The second bullet point above is prove that *problem B* is NP-hard which combined with the first bullet point yields the NP-complete proof.

Problem 1
(15 points)

Show that each of the following problems is in NP. For each problem, assume the witness gives you a candidate solution S. **You must show how to verify S in polynomial time (and not simply just solve the problem!)**.

(a) Given an two strings X and Y of length n and m respectively, determine if there exists a common subsequence between X and Y of length $\geq d$.

Solution: Solution: Suppose the witness provides a common subsequence S of length $\geq d$. Check that solution is indeed a subsequence of both X and Y, done by scanning through each string to confirm that each character in solution appears in the same order. Taking O(n+m) time which is polynomial.

(b) Given a undirected, weighted graph $G = (V, E)$, does there exist an spanning tree of total weight $\leq b$?

Solution: Suppose the witness provides a spanning tree T of G, verifying that T has V-1 edges and calculate the total weight of T by summing the weights of its edges. Verify that the total weight is $\leq b$. which can be done in $O(|V| + |E|)$ time.

(c) Given a list of sets $\{S_1, S_2, \ldots, S_n\}$ and a budget b, does there exist a set H of size $\leq b$ which intersects every S_i?

Solution: Suppose the witness provides a set H of size $\leq b$. Verify that $|H| \leq b$. For each set S_i, check that $H \cap S_i \neq \emptyset$, ensuring H intersects every S_i. These checks can be done in polynomial time relative to the size of the input. Suppose H has up to b elements and each S_i has at most m elements. Checking if H intersects S_i (by scanning for at least one common element) takes $O(b \times m)$ time for each S_i. Repeating this check for each of the n sets gives a total runtime of $O(n \times b \times m)$.

Problem 2
(40 points)

Answer the following questions using proofs when required or explanations:

(a) Prove that polynomial-time reductions are transitive. That is, if $A \leq_p B$, and $B \leq_p C$, then $A \leq_p C$.

Solution: To show that polynomial-time reductions are transitive, let's assume: $A \leq_p B$, meaning there exists a polynomial-time reduction f from A to B.
$B \leq_p C$, meaning there exists a polynomial-time reduction g from B to C.
Now, to show that $A \leq_p C$:
1. For any instance x of A, apply reduction f to get an instance $y = f(x)$ of B
2. Then apply the reduction g to y to obtain an instance $z = g(f(x))$ of C.
3. Since both f and g run in polynomial time, the composition $h(x) = g(f(x))$ also runs in polynomial time (since the composition of two polynomial functions is also polynomial).
Therefore, $A \leq_p C$, proving that polynomial-time reductions are transitive.

(b) Prove that `2-SAT` $\leq_p$ `3-SAT`

Solution:

In 2-SAT, each clause has at most two literals (e.g., $(x \vee y)$ or $(\neg x \vee y)$).

To convert a 2-SAT clause $(x \vee y)$ into a 3-SAT form, add an extra literal, say z, such that

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z)$$

represents the same logic as $(x \vee y)$. This way, each 2-literal clause is transformed into two 3-literal clauses. Thus, we can convert any 2-SAT instance into an equivalent 3-SAT instance in polynomial time, proving that 2-SAT $\leq_p$ 3-SAT.

(c) Prove that if `3-SAT` $\leq_p$ `2-SAT` then P = NP.

Solution:

3-SAT is a known NP-complete problem, meaning that if we could reduce 3-SAT to 2-SAT in polynomial time, it would imply that 2-SAT is also NP-complete.

However, 2-SAT is known to be solvable in polynomial time (it's in **P**).

If 3-SAT (an NP-complete problem) could be reduced to a polynomial-time solvable problem, it would mean that all NP problems could also be solved in polynomial time, thus proving $P = NP$. Therefore, if 3-SAT $\leq_p$ 2-SAT, then $P = NP$.

(d) Show that the following `DNF-SAT` problem is in class P.

Input: A boolean formula in Disjunctive Normal Form with m clauses and n variables.
Example Input: $f(x) = (x_1 \wedge x_2 \wedge \overline{x}_1) \vee (x_3 \wedge x_2 \wedge \overline{x}_2) \vee (x_3 \wedge x_2 \wedge \overline{x}_1)$

Output: Determines whether there exists a satisfying assignment such that the boolean formula evaluates to True.

Solution:

In DNF-SAT (Disjunctive Normal Form SAT), the formula is in a disjunction of conjunctions, such as

$$(x_1 \wedge x_2 \wedge \neg x_3) \vee (x_3 \wedge x_2 \wedge \neg x_2).$$

A DNF formula is satisfied if at least one of its conjunctions (terms) is satisfiable.

To verify if the formula is satisfiable, we can simply check each conjunction independently to see if there exists an assignment that makes it true.

Each conjunction in DNF is satisfied by a specific assignment of variables, so checking each conjunction takes polynomial time in the number of clauses and variables. Since we only need to find one satisfying conjunction, the verification process is in P. Thus, DNF-SAT is in class P.

Problem 3
(22.5 points)

Show that the following `TwoThirdsMajority-SAT` problem is NP- Complete.

Input: a boolean formula in CNF with n variables and m clauses, where m is divisible by 3.
Output: Whether or not a satisfying assignment exists such that at least 2/3 of the clauses in the given input CNF evaluates to True.

Note: Write the reduction in the format described on the front page of this document.

Solution: **Show that TwoThirdsMajority-SAT is in NP**

Witness: A potential solution (assignment of variables) that we can verify.

Verification: Given an assignment of the variables, check how many clauses in the CNF formula evaluate to true.

Counting Clauses: Count the number of clauses that are satisfied under this assignment. This can be done in $O(m)$, where m is the number of clauses.

Check Condition: Verify that at least $\frac{2}{3}m$ of the clauses are satisfied. Since this verification process runs in polynomial time, TwoThirdsMajority-SAT is in NP.

Show that TwoThirdsMajority-SAT is NP-Hard

Input: A 3-SAT formula in CNF with m clauses and n variables.

Transformation:

For a given 3-SAT formula ϕ, create an instance of TwoThirdsMajority-SAT by using the same formula.

Set the requirement that at least $\frac{2}{3}m$ of the clauses need to be satisfied.

Correctness of the Reduction:

If there exists a satisfying assignment for the original 3-SAT formula, then all m clauses can be satisfied. This assignment would also satisfy at least $\frac{2}{3}m$ clauses in the TwoThirdsMajority-SAT instance.

Conversely, if a TwoThirdsMajority-SAT instance with the transformed formula has a satisfying assignment that satisfies at least $\frac{2}{3}m$ clauses, then it implies that a significant number of clauses are satisfiable, making it a candidate for satisfying the original 3-SAT formula.

Polynomial-Time Conversion: The transformation from 3-SAT to TwoThirdsMajority-SAT does not increase the problem size significantly and can be done in polynomial time.

Since we have shown that TwoThirdsMajority-SAT is in NP and that a polynomial-time reduction exists from 3-SAT (an NP-complete problem) to TwoThirdsMajority-SAT, we conclude that TwoThirdsMajority-SAT is NP-complete.

Problem 4
(22.5 points)

Show that the following `Integer-Inequalities` problem is NP-Complete through a reduction from `3-SAT`.

Input: A system of m linear inequalities with n variables $x_1,\ dots, x_n$ where each inequality is in the form

$$\mathbf{a}^T\mathbf{x} \leq b$$
$$\text{s.t } \mathbf{a} \in \mathbb{R}^n, \mathbf{x} = [x_1, \ldots, x_n]^T, b\ in\mathbb{R}$$

Output: Whether or not an assignment of the n variables to textbfinteger values exists such that each of the m inequalities are satisfied.
Example: Given these inequalities with variables x_1, x_2:

$$-x_1 - 2x_2 \leq -3$$
$$2x_1 - x_2 \leq 4$$
$$-x_1 + x_2 \leq 1$$

assigning $x_1 = 2$ and $x_2 = 1$ would satisfy these inequalites.
Note: Write the reduction in the format described on the front page of this document.

Solution:

Show that Integer-Inequalities is in NP

Witness: A potential integer assignment for each variable $x_1, x_2, \ldots, x_n$.

Verification: Given an assignment of integer values to each variable, check each inequality to verify that it is satisfied.

Verification Process:

For each inequality $\mathbf{a}^T\mathbf{x} \leq b$, compute $\mathbf{a}^T\mathbf{x}$ using the values in the assignment.

Compare the result with b to ensure it satisfies the inequality.

Since evaluating each inequality is a linear operation, verifying all m inequalities can be done in $O(m \cdot n)$, which is polynomial in the size of the input. Since this verification process runs in polynomial time, the Integer-Inequalities problem is in NP.

Show that Integer-Inequalities is NP-Hard

Input: A 3-SAT formula in CNF with m clauses and n variables, where each clause has exactly three literals (e.g., $(x_1 \vee \neg x_2 \vee x_3)$).

Constructing the Integer-Inequalities System:

For each variable x_i in the 3-SAT formula, introduce a corresponding integer variable y_i in

the Integer-Inequalities system.

Encode each clause in the 3-SAT formula as a set of inequalities that enforce at least one of the literals in each clause to be true.

For example, consider a clause $(x_1 \vee \neg x_2 \vee x_3)$. We can represent this clause by introducing constraints that force at least one of the following conditions to hold:

$y_1 \geq 1$ (if x_1 is true),
$y_2 \leq 0$ (if $\neg x_2$ is true),
$y_3 \geq 1$ (if x_3 is true).

We combine these inequalities to represent the clause as a system of linear inequalities.

Ensuring Satisfiability:

Each clause in the 3-SAT formula contributes a set of inequalities in the Integer-Inequalities system that enforces that at least one literal in each clause is satisfied.

By constructing the inequalities in this manner, any satisfying assignment to the 3-SAT formula will correspond to an integer solution to the Integer-Inequalities system, and vice versa.

Polynomial-Time Conversion:

The transformation from a 3-SAT formula to an Integer-Inequalities system involves creating inequalities for each variable and clause, which can be done in polynomial time.