CIS 3200: Algorithms Spring 2024

Computational Hard Problems

Author: Sophie :)

1 Definitions

P: The set of problems solvable in polynomial time. This means for a problem of size n, it can be solved in $n^{O(1)}$ time.

NP: The set of decision problems solvable in nondeterministic polynomial time. In other words:

- A decision problem must have a YES or NO answer.
- Nondeterministic means we can guess a solution out of a polynomial number of options in $O(1)$ time.
- If YES is a possible answer, a nondeterministic algorithm will make that guess.

NP-hard: A problem X is NP-hard if there exists a polynomial-time reduction from every problem Y in NP to X.

Reductions and Reducibility: Given two decision problems X and Y, Y is reducible to X (denoted $Y \leq_p X$) if there exists a polynomial-time computable function f such that for every instance y of problem Y, y is a YES-instance of Y if and only if $f(y)$ is a YES-instance of X. In other words, Y can be efficiently transformed into X such that the solution to Y corresponds to the solution of X.

What does this mean?

X Solves Y: If we can use X to solve Y efficiently, it implies that X is at least as hard as Y. This means that if Y is computationally challenging, X must be at least as challenging, if not more so.

NP-hardness Implication: If Y is NP-hard, it means that every problem in NP can be reduced to Y in polynomial time. If X can solve Y, and Y is NP-hard, then X must also be at least as hard as Y. In other words, X must also be NP-hard.

NP-complete: A problem X is NP-complete if $X \in$ NP and X is NP-hard.

2 Proving NP-Completeness

Proving NP-Completeness involves three steps:

1. Define the decision question with an input and a goal.
2. Prove $X \in$ NP. Provide:

   (a) Certificate: A solution to be verified.
   (b) Verifier: An algorithm that checks the validity of the certificate in polynomial time.

3. Prove X is NP-Hard. This includes:
   (a) Reduction from a known NP-hard problem Y: Provide a polynomial-time transformation from Y inputs to X inputs.
   (b) Proof of correctness: Show that if Y answers YES, X answers YES, and vice versa.

Since $X \in$ NP and X is NP-hard, X is NP-complete.

3 Some Famous Examples

Let's take a look at some examples of NP-Complete problems and how we would go about proving their NP-Completeness.

3.1 3SAT

Decision Question:

Input: A boolean formula of the form: $(x_1 \vee x_3 \vee \bar{x_6}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_7}) \wedge ...$

Goal: Determine an assignment of variables to True and False such that the whole formula evaluates to True.

Proof 3-SAT $\in$ NP:

Certificate: A list of assignments for each literal.

Verifier: Evaluate the formula with the variable assignments. This can be done in polynomial time.

Proof 3-SAT is NP-hard:

Since this is our first NP-complete problem, we can't write a reduction from another NP-complete problem! I challenge you to go back after reading about another NP-complete problem and attempt to write a reduction for 3-SAT.

Instead, we will take for granted for now that 3-SAT is NP-hard. Look up Cook's Theorem if you want to learn more about how this was proven.

In sum: Since 3-SAT $\in$ NP and 3-SAT is NP-hard, 3-SAT is NP-complete.

3.2 Independent Set

Decision Question:

Input: An undirected graph $G = (V, E)$ and an integer k.

Goal: Determine whether there exists a subset $V' \subseteq V$ with $|V'| = k$ such that no two vertices in V' are adjacent to each other.

Proof Independent Set $\in$ NP:

Certificate: A set of k vertices, V'.

Verifier: For each pair of vertices $u, v \in V'$, we check that there is no edge between them. This can be done in polynomial time, especially if we assume an adjacency matrix representation of the graph.

Proof Independent Set is NP-hard:

Now that we have a known NP-complete problem, 3-SAT, we can reduce this to Independent Set to prove that Independent Set is NP-hard. Let's break down this reduction step:

Reduction from 3-SAT:

We aim to represent the inputs to 3-SAT as inputs to Independent Set such that the requirements for both problems are maintained.

Inputs to 3-SAT include literals (x_i and $\bar{x_i}$) and clauses ($C_1, C_2, \ldots$). We will map literals to vertices in the Independent Set problem.

Now, let's consider the constraints imposed by 3-SAT and how to translate them to Independent Set:

- If x_i is True, $\bar{x_i}$ must be False. To handle this, we draw an edge between each x_i and $\bar{x_i}$ in the Independent Set graph, ensuring that only one of them can be in the independent set.
- Each clause in 3-SAT must have at least one literal assigned True. In the Independent Set problem, this translates to drawing edges between the vertices corresponding to literals in the same clause, ensuring that at least one of them is in the independent set.

We set k equal to the number of clauses in 3-SAT and solve for Independent Set.

Proof of Correctness:

- $\Rightarrow$ If 3-SAT is satisfiable, then each clause has at least one true literal. We select one literal from each clause to remain true, resulting in k literals total. Let these k literals be the members of V' in the independent set. Since each literal is connected to its negation and to other literals in the same clause, no vertex in V' will be connected to another vertex in V'. Thus, Independent Set is satisfiable.
- $\Leftarrow$ If Independent Set is satisfiable, we have a set of vertices V' where no two vertices are adjacent. This implies that no two literals in V' are conflicting, ensuring that at least one literal from each clause is assigned True. Hence, there exists a valid assignment of literals to solve 3-SAT.

In summary: Since Independent Set $\in$ NP and Independent Set is NP-hard, Independent Set is NP-complete.