Complexity Classes and NP-completeness

1 Complexity Classes

1.1 Polynomial Time

The class **P** contains all problems which can be solved in polynomial time. Formally, **P** contains all problems X such that there exists some algorithm A_X which always finishes in $P_{A_X}(n)$ steps (where P_{A_X} is a polynomial) on inputs of size n, and correctly solves X.

1.2 Nondeterministic Polynomial Time

An **efficient certifier** B is a program with a proposed solution s and certificate t such that B runs in polynomial time $T_B(|s|, |t|)$. It is a certifier for a problem X if the following holds:

- $s \notin X \implies B(s, t)$ answers "No" for all t.
- $s \in X \implies$ there exists a polynomial length t such that $B(s, t)$ answers "Yes".

The class **NP** contains all problems with efficient certifiers. It is simple to show that **P** $\leq$ **NP** because a polytime solution algorithm with a null certificate is an efficient verifier.

It is an open question (perhaps the greatest open question in all of computer science) if **P** $=$ **NP**. In other words, if it is easy to verify a solution, is it also easy to find a solution? **NP** is one of the most important complexity classes because most problems computer scientists care about are in **NP**. That's why we focus on it, and why finding out if **P** $=$ **NP** is so important.

Being in **NP** allows exhaustive search over candidate certificates, then running the certifier on each, to guarantee a runtime upper bound of $\mathcal{O}(2^{P(n)}T_B(n))$ and polynomial space. Thus, it can be shown that **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME.**

1.3 Co-nondeterministic Polynomial Time

The class **co-NP** contains all problems which efficient "No" certifiers (there exists a polynomial time algorithm which can verify *no* counterexamples given the appropriate certificate t). In other words, **co-NP** contains all problems X such that $\bar{X} \in$ **NP**. It is an open question if **NP** $=$ **co-NP**.

1.4 Karp Reductions

We are motivated to study the hardest problems in **NP**, but the definition of "hard" can be rather elusive. In computational complexity, we say that a problem X is harder than Y if $Y \leq pX$. That is, X is harder if X can be reduced to Y.

A Karp reduction A from X to Y is a polytime algorithm such that:

- $x \in X \iff A(x) \in Y$ ("Yes" maps to "Yes": completeness)
- $x \notin X \iff A(x) \notin Y$ ("No" maps to "No": soundness)

Karp reductions are the main tool in proving whether a problem is a member of a certain complexity class.

2 NP-hardness and NP-completeness

2.1 Definitions

A problem X is **NP**-hard if $Y \leq pX \; \forall Y \in \mathbf{NP}$.

A problem X is **NP**-complete if X is **NP**-hard and $X \in \mathbf{NP}$.

2.2 The Cook-Levin Theorem

Cook and Levin proved in the early 1970s that 3SAT is **NP**-complete – the first problem to be confirmed as such. 3SAT stands for three-boolean satisfiability, and asks for a solution to the formula $F = \bigwedge_{j=1}^{m} C_j$, where each $C_j = l_{j_1} \vee l_{j_2} \vee l_{j_3}$, and each $l_{j_i}$is a boolean variable. This theorem had several implications: **NP**-complete problems exist, and 3SAT is generic enough that it can easily reduce to other important problems.

2.3 Proving NP-completeness

Claim 1. Karp reduction is transitive (i.e, $Z \leq pX \leq pY \implies Z \leq pY$).

Proof. We have polytime algorithms A_Z reducing Z to X and A_X reducing X to Y. To obtain a polytime algorithm A reducing Z to Y, we simply run A_X on the output of A_Z, since two subsequent polytime algorithms (where the output of the first has polynomial size) still runs in polytime.

$$z \in Z \iff A_Z(z) \in X \iff A_X(A_Z(z)) \in Y \iff A(z) \in Y$$

□

Proposition 2. *If X is **NP**-hard, $Y \in$ **NP**, $X \leq pY$, then Y is **NP**-complete.*

Proof. We know $Z \leq pX \; \forall Z \in \mathbf{NP}$ and $X \leq pY$, so by Claim 1, $Z \leq pY \; \forall Z \in \mathbf{NP}$. Since $Y \in \mathbf{NP}$, this is the definition of **NP**-complete. □

2.4 CERT is NP-complete

It's rather difficult to prove the Cook-Levin Theorem (which is why they won a Turing award for it), but we can provide an example on a simpler, yet less useful problem.

Let CERTIFICATION (CERT) is the following problem: Given a certifier algorithm B with time bound T_B, certificate length P_B, and input s, is there a certificate t with $|t| \leq P_B(|s|)$ such that $B(s,t)$ answers "Yes" after at most $T_B(|s|,|t|)$ steps?

Theorem 3. *CERT is **NP**-complete.*

Proof. A two-part proof: first that CERT $\in$ **NP**, then that CERT is **NP**-hard.

(1). CERT $\in$ **NP**:

Given an input (B, T_B, P_B, s) and proposed certificate t, we check if $|t| \le P_B(|s|)$, then simulate $B(s,t)$ for $T_B(|s|, |t|)$ steps. If both succeed, we answer "Yes".

(2). CERT is **NP**-hard:

Proposition 4. *$X \le p$ CERT $\forall X \in$* ***NP***

Proof. Because $X \in$ **NP**, it has an efficient certifier B_X with certificate length P_{B_X} and runtime T_{B_X}. Given an input string s, our reduction outputs B_X, T_{B_X}, P_{B_X}, and s. Then:

$$s \in X \iff \exists t : |t| \le P_{B_X}(s) \text{ and } B(s,t) \text{ answers "Yes" in } T_{B_X}(|s|,|t|) \text{ steps} \iff (B_X, T_{B_X}, P_{B_X}, s) \in \text{ CERT}$$

□

Proposition 4 is the definition of **NP**-hardness, so CERT is proven **NP**-hard. Since it satisfies both conditions, CERT is **NP**-complete. □