Problem Reduction Sequence

Any NP problem

CIRCUIT-SAT

SAT

3-CNF-SAT

3SAT

3SAT ≤P CLIQUE

VERTEX-COVER

CLIQUE ≤P CLIQUE

SUBSET-SUM

3SAT ≤P CLIQUE

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

TSP

All of these are NP-Complete

last section

Chapter 3

CS 3343 Analysis of Algorithms

NP-Completeness – 5
A concrete problem is \textit{polynomial-time solvable}, therefore, if there exists an algorithm to solve it in time $O(n^k)$ for some constant $k$.

We can now formally define the \textit{complexity class} $\mathbf{P}$ as the set of concrete decision problems that are polynomial-time solvable.

Using this language-theoretic framework, we can provide an alternative definition of the complexity class $\mathbf{P}$:

\[
P = \{ L \subseteq \{0, 1\}^* : \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time} \}.
\]

$\text{Bellman-Ford} \quad \Theta(VE) \quad O(n^2)$

$\text{Bubble Sort} \quad \Theta(n^2)$
Verification Algorithms

We define a verification algorithm as being a two-argument algorithm \( A \), where one argument is an ordinary input string \( x \) and the other is a binary string \( y \) called a certificate. A two-argument algorithm \( A \) verifies an input string \( x \) if there exists a certificate \( y \) such that \( A(x, y) = 1 \). The language verified by a verification algorithm \( A \) is

\[
L = \{ x \in \{0, 1 \}^* : \text{there exists } y \in \{0, 1 \}^* \text{ such that } A(x, y) = 1 \}.
\]

Idea — many algs have a set/sequence of choices to make:

- 3SAT — set of choices what values to assign Boolean vars
- TS — sequence of choices to form a circuit

Polynomial-Time Verification Algorithm

Can I verify if a set/sequence of choices is correct?

3SAT — \( \Theta \) can verify if an assignment makes \( \Phi \) true?
TS — \( \Theta \) can verify if a circuits is \( \triangle \) distance?
The complexity class $\text{NP}$ is the class of languages that can be verified by a polynomial-time algorithm. More precisely, a language $L$ belongs to NP if and only if there exist a two-input polynomial-time algorithm $A$ and a constant $c$ such that

$$L = \{x \in \{0, 1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}.$$ 

Computer scientists believe $P \neq \text{NP}$, but no proof exists.
Returning to our formal-language framework for decision problems, we say that a language $L_1$ is **polynomial-time reducible** to a language $L_2$, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$,

\[
x \in L_1 \text{ if and only if } f(x) \in L_2.
\]  

\[\text{convert instances of } A \]

\[\text{into instances of } B \]

\[\text{so that the answer for } B \]

\[\text{is also the answer for } A \]

\[\text{(34.1)}\]
A language \( L \subseteq \{0, 1\}^* \) is **NP-complete** if

1. \( L \in \text{NP} \), and
2. \( L' \leq_p L \) for every \( L' \in \text{NP} \).—can convert every \( \text{NP} \) problem to \( L \)

If a language \( L \) satisfies property 2, but not necessarily property 1, we say that \( L \) is **NP-hard**. We also define \( \text{NPC} \) to be the class of NP-complete languages.

As the following theorem shows, NP-completeness is at the crux of deciding whether \( \text{P} \) is in fact equal to \( \text{NP} \).

**Theorem 34.4**

If any NP-complete problem is polynomial-time solvable, then \( \text{P} = \text{NP} \). Equivalently, if any problem in \( \text{NP} \) is not polynomial-time solvable, then no NP-complete problem is polynomial-time solvable.
How computer scientists think problems are structured.

NP-hard

NPC

P

NP

Identify factors of integers.

Is a number composite?