NP-Complete Problems

Complexity Class P

Abstract Problems. An *abstract problem* Q is a binary relation on sets I of *input instances* and S of *problem solutions*.

$Q(i, s)$, means that s is a solution for the abstract problem Q given input i.

Decision problems. An abstract problem Q is a *decision problem* if $S = \{0, 1\}$ (or $\{\mathsf{false}, \mathsf{true}\}$).

Decision problems determine whether something is *possible* for a given input.

Example. The following are all decision problems:

- Given a directed graph and two nodes x and y in it, is there a path from x to y?
- Given an array of numbers A and a number x, is it true that x is the smallest number in A?
- Is a given undirected graph G connected?
- Given two strings s_1 and s_2, is there an alignment for them with an edit distance of no more than a given number k?

Encodings. Let S be a set of *abstract* objects. An encoding of S is a mapping from elements of S onto a set of *binary strings*.

Example. Some examples of encodings:

- The set $\{\mathsf{false}, \mathsf{true}\}$ can be encoded as $\{0, 1\}$.
- The set $\mathcal{N}$ of natural numbers can be encoded as $\{0, 1, 10, 11, 100, 101, \ldots\}$.
- A graph $G = \{V, G\}$ can be encoded as a concatenation of two strings $e(V)$ of length $log_2(|V|)$ and $e(E)$ of length $|V|^2$.

  $e(V)$ is a binary encoding of the natural number $|V|$.

  $e(E)$ is a concatenation of $|V|$ bitmaps $e(v_i)$. $e(v_i)$ has a 1 in position j, if $(v_i, v_j) \in E$ and 0 in position j otherwise.

Concrete problems. A *concrete problem* is a problem whose input and output sets are *binary strings.*

Computational complexity of concrete problems. A concrete problem is solvable in time $O(T(n))$ iff when there exists an algorithm which, when provided an instance i of the problem of size $|i| = n$, solves the problem instance in time $O(T(n))$.

Polynomial complexity. A concrete problem is *polynomially solvable* iff it is solvable in $O(n^k)$ time for some constant k.

Complexity of abstract problems. A function $f : \{0, 1\}^* \to \{0, 1\}^*$ is *polynomial-time computable* iff there exists an algorithm A which given an input binary string i produces $f(i)$ in $O(n^k)$ time.

Two *encodings* e_1 and e_2 are *polynomially related* if there are polynomially computable functions f_{12} and f_{21} such that

$$f_{12}(e_1(i)) = e_2(i)$$

and

$$f_{21}(e_2(i)) = e_1(i).$$

Formal language framework. A an *alphabet* Σ is a finite set $\Sigma = \{s_1, \ldots, s_m\}$ of symbols. A *string* $S \in \Sigma^*$ is a sequence $x_1 x_2 \ldots x_n$ of symbols, such that $x_i \in \Sigma$.

A *formal language* in alphabet Σ is a set $L = \{l_1 \ldots\}$ of strings from alphabet Σ.

An algorithm A ***accepts*** a string $x \in \{0, 1\}^*$ iff, given x as input, it outputs 1, i.e., if $A(x) = 1$. An algorithm A *accepts* a language L iff $A(x) = 1$ for each $x \in L$.

An algorithm A **decides** a language L iff:

- $A(x) = 1$ for each $x \in L$;
- $A(x) = 0$ for each $x \notin L$.

An algorithm A *accepts* L *in polynomial time* iff

- A accepts L.

- There exists a constant k, such that on any string $x \in L$, $|x| = n$, A accepts x in time $O(n^k)$.

An algorithm A *decides* L *in polynomial time* iff

- A decides L.
- There exists a constant k, such that on any string $x \in \{0,1\}^*$, $|x| = n$, A decides x in time $O(n^k)$.

Complexity class. A **complexity class** is a set of languages membership in which is determined by a **complexity measure** of an algorithm deciding the language.

Complexity class P. Complexity class **P** is defined as

$\mathbf{P} = \{L \subseteq \{0,1\}^* |$ there exists an algorithm A that decides L in ***polynomial time***$\}$.

Theorem. $P = \{L|\ L$ is *accepted* in polynomial time$\}$.

Complexity Class NP

Verification. A **verification algorithm** A for is an algorithm that takes as input two strings, x and y (usually called a *certificate* and produces as output a $\{0,1\}$ decision.

A **verifies** x iff *there exists a certificate* y, such that $A(x,y) = 1$.

A language L is **verified** by A is defined as

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

Complexity class NP. The ***complexity class*** **NP** is the class of languages that can be *verified by a polynomial algorithm.* More formally, a language L belongs to class **NP** iff there exist an two-input polynomial algorithm A and a constant c such that

$$L = \{x \in \{0,1\}^* |\text{ there exists } y \in \{0,1\}^* \text{ s.t. } |y| \leq O(|x|^c) \text{ and } A(x,y) = 1\}.$$

Complexity class $\mathbf{co-NP}$. The complexity class $\mathbf{co-NP}$ is defined as $=\{L|\bar{L} \in \mathbf{NP}\}$.

NP-Completeness

Some problems are harder than others. *NP-Complete* problems are the problems in class **NP** that are as hard to solve as any other problem in that class.

This notion can be formalized.

Reducability. A langauge L_1 is ***polynomial-time reducible*** to language L_2, denoted $L_1 \leq_p L_2$ iff, there exists a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$:

$$x \in L_1 \textbf{iff} f(x) \in L_2.$$

f is called the *reduction function.*

NP-Completeness. A language $L \subset \{0,1\}^*$ is **NP-complete** iff

- $L \in \mathbf{NP}$
- $L' \leq_p L$, for **every** $L \in \mathbf{NP}$.

NP-hardness. A language $L \subset \{0,1\}^*$ is **NP-complete** iff

- $L' \leq_p L$, for **every** $L \in \mathbf{NP}$.

Big Questions

Are there NP-complete problems? Yes, *there are!*

What is the relationship between P and NP? We observe:

Theorem. $\mathbf{P} \subseteq \mathbf{NP}$.

This is so, because a polynomial algorithm that **actually solves** the problem, will also *verify* the solution.

Theorem. If **any NP-complete** problem is polynomial-time solvable, then $\mathbf{P} = \mathbf{NP}$.

If **any NP-complete** problem is NOT polynomial-time solvable, then $\mathbf{P} \subset \mathbf{NP}$.

The biggest unsolved mystery in Computer Science. Is $\mathbf{P} = \mathbf{NP}$?

Note. This problem actually has many important practical implications. E.g., if $P = NP$, then a lot of encryption algorithms are obviated and rendered ineffective in protecting information. That is, a lot of current practical work relies on *NP-complete* problems being hard to solve.

NP-complete Problems: Examples

Hamiltonian Cycle. Given a graph $G = \{V, E\}$, a **hamiltonial cycle** of G is a path p that starts and ends with the same node (a cycle) and visits every other node **exactly** once.

Satisfiability. Given a propositional boolean formula consisting of propositional variables, and connectives $\wedge, \vee, \neg$, is there an assignment of truth values to the propositional variables that make the formula **true**?

3-CNF satisfiability. Given a collection disjunstions $D_1, \ldots, D_k$ of the form $l_1 \vee l_2 \vee l_3$ where l_1 is either some propositional variable x or its negation $\neg x$, determine if $C = D_1 \wedge D_2 \wedge \ldots \wedge D_k$ is satisfiable.

Longest simple path. Given an edge-weighted graph $G = (V, E, w)$ find the *longest simple path* in G.

Clique. Given an undirected graph $G = (V, E)$, a **clique** is a subset $V' \subseteq V$ of vertices, such that there is an edge between each pair. $|V'|$ is the size of the clique. A decision problem is, given a graph G and an integer k determine if G has a clique of size k.