The Design and Analysis of Algorithms

Lecture 16 Intractability I

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Algorithm Design Patterns and Anti-Patterns

- *Algorithm design patterns*.

  Greed.

  Divide-and-conquer.

  Dynamic programming.

  Reductions.

  Local search, randomization $\cdots$

- *Algorithm design anti-patterns*.

  NP-completeness.

  PSPACE-completeness.

  Undecidability.

Computational Models–Turing Machine

- The Turing machine (TM) is a mathematical model of computation, it has

1. A memory tape with an infinite line of cells, each of which contains a symbol from a finite alphabet Γ.
2. A finite number of possible states.
3. A read/write head, which at each step can read/write one cell of the tape and move one step left or right.
4. A *transition* function determines what to do when it is in a particular state and the head reads a particular symbol.

- We usually use its extension: multi–tape Turing machines.

Turing machine

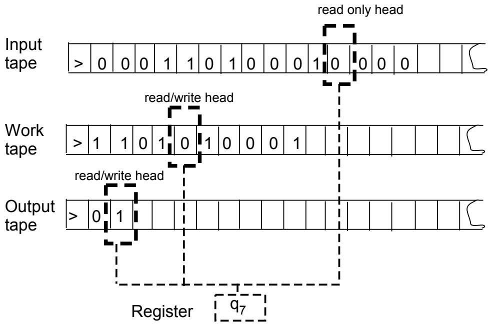
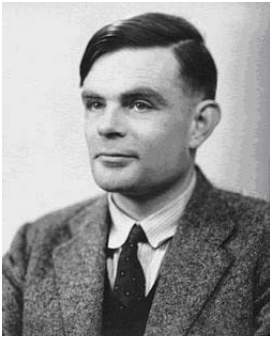


Figure 1: A single step by a Turing machine

The Church–Turing Thesis

A function can be computed by some Turing machine if and only if it can be computed by some machine of any other "reasonable and general" model of computation.



Classify Problems Via Computational Requirements

Q. Which problems will we be able to solve in practice?

A. Those with polynomial-time algorithms.

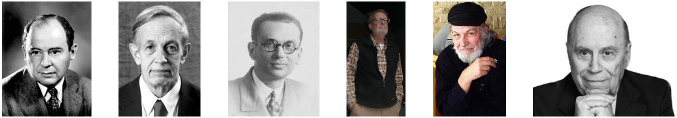


Figure 2: von Neumann (1953), Nash (1955), Gödel (1956), Cobham (1964), Edmonds (1965) and Rabin (1966)

Theory. Definition is broad and robust.

Practice. Poly-time algorithms scale to huge problems.

Decision Problems

- *Decision problem.*

  Problem X is a set of strings.

  Instance s is one string.

  Algorithm A solves problem X: $A(s) = yes$ iff $s \in X$.

Def. Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial.

Ex. Problem $PRIMES = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \cdots\}$.

Instance $s = 592335744548702854681$.

AKS algorithm *PRIMES* in $O(|s|^8)$ steps.

Definition of P

P. Decision problems for which there is a poly-time algorithm.

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is x a multiple of y?	Grade school division	51, 17	51, 16
RELPRIME	Are x and y relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is x prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5?	Dynamic programming	niether neither	acgggt ttttta
LSOLVE	Is there a vector x that satisfies Ax = b?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

NP

- *Certification algorithm intuition.*

  Certifier views things from "managerial" viewpoint.

  Certifier doesn't determine whether $s \in X$ on its own;

  It checks a proposed proof t that $s \in X$.

Def. Algorithm $C(s, t)$ is a certifier for problem X if for every string s, $s \in X$ iff there exists a string t such that $C(s, t) = yes$.

Def. *NP* is the set of problems for which there exists a poly-time certifier.

$C(s, t)$ is a poly-time algorithm.

Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial p.

Remark. *NP* stands for *nondeterministic* polynomial time.

Certifiers and Certificates: Composite

- *COMPOSITES*. Given an integer s, is s composite?
- *Certificate*. A nontrivial factor t of s. Such a certificate exists iff s is composite. Moreover $|t| \leq |s|$.
- *Certifier*. Check that $1 < t < s$ and that s is a multiple of t.

Ex. Instance s: 437669; certificate t: 541 or 809.

- *Conclusion*. *COMPOSITES* is in *NP*.

Constraint Satisfaction Problems

- *Literal*. A boolean variable or its negation, x_i or $\bar{x}_i$.
- *Clause*. A disjunction of literals, $C_j = x_1 \vee \bar{x}_2 \vee x_3$.
- *Conjunctive normal form*. A propositional formula Φ that is the conjunction of clauses, $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$.
- *SAT*. Given CNF formula Φ, does it have a satisfying truth assignment?
- *3-SAT*. SAT where each clause contains exactly 3 literals.

Ex. $\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$.

- Yes instance: $x_1 = \mathit{true}, x_2 = \mathit{true}, x_3 = \mathit{false}, x_4 = \mathit{false}$.

Certifiers and Certificates: 3-SAT

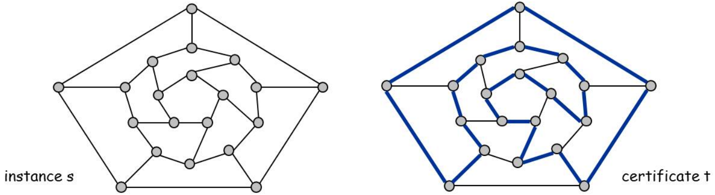
- *3-SAT*. Given a CNF formula Φ, is there a satisfying assignment?
- *Certificate*. An assignment of truth values to the n boolean variables.
- *Certifier*. Check that each clause in Φ has at least one true literal.

Ex. Instance $s : \Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$.

- *Certificate* $t : x_1 = \mathit{true}, x_2 = \mathit{true}, x_3 = \mathit{false}, x_4 = \mathit{false}$.
- *Conclusion*. *3-SAT* is in *NP*.

Certifiers and Certificates: Hamiltonian Cycle

- *HAM-PATH*. Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every node?
- *Certificate*. A permutation of the n nodes.



- *Certifier*. Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes.
- *Conclusion*. *HAM-PATH* is in *NP*.

P and NP

P. Decision problems for which there is a poly-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

EXP. Decision problems for which there is an exponential-time algorithm.

Claim. $P \subseteq NP$.

Pf. Consider any problem $X \in P$.

By definition, there exists a poly-time algorithm $A(s)$ that solves X.

Certificate $t = \emptyset$, certifier $C(s, t) = A(s)$. $\square$

P, NP, and EXP

Claim. $NP \subseteq EXP$.

Pf. Consider any problem $X \in NP$.

By definition, there exists a poly-time certifier $C(s, t)$ for X.

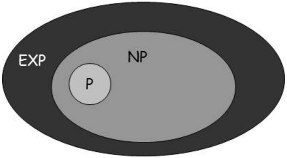
To solve input s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$. $\square$

Remark. Time-hierarchy theorem implies $P \subsetneq EXP$.

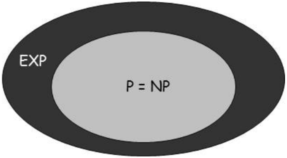
The Main Question: P Versus NP

- Does $P = NP$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

  Is the decision problem as easy as the certification problem?
- *If yes*. Efficient algorithms for *3-SAT*, *TSP*, *3-COLOR*, *FACTOR*, $\cdots$
- *If no*. No efficient algorithms possible for *3-SAT*, *TSP*, *3-COLOR*, *TSP*, $\cdots$
- *Consensus opinion*. Probably no.
- *Millennium prize*. \$ 1 million for resolution of $P = NP$ problem.



If $P \neq NP$



If $P = NP$

Polynomial Reduction

Def. Problem X *polynomial (Cook) reduces* to problem Y if arbitrary instances of problem X can be solved using:

Polynomial number of standard computational steps, plus

Polynomial number of calls to oracle that solves problem Y.

Notation. $X \leq_P Y$.

Def. Problem X *polynomial (Karp) transforms* to problem Y if given any input x to X, we can construct an input y such that x is a *yes* instance of X iff y is a *yes* instance of Y.

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all forthcoming reductions are of this form.

- *Open question*. Are these two concepts the same with respect to *NP*?

Homework

- Read Chapter 8 of the textbook.
- Exercises 6 & 23 in Chapter 8.