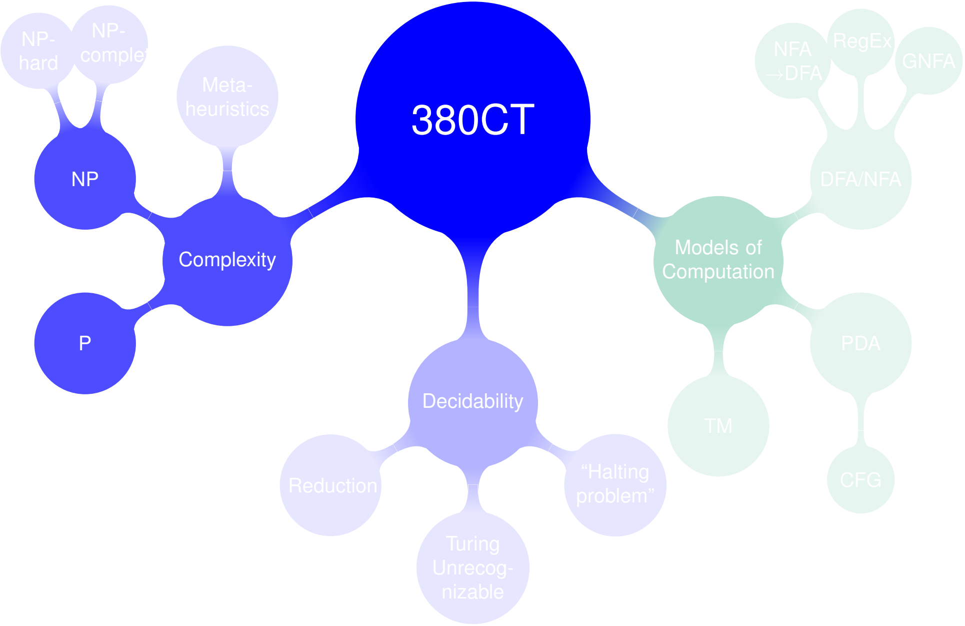
Complexity

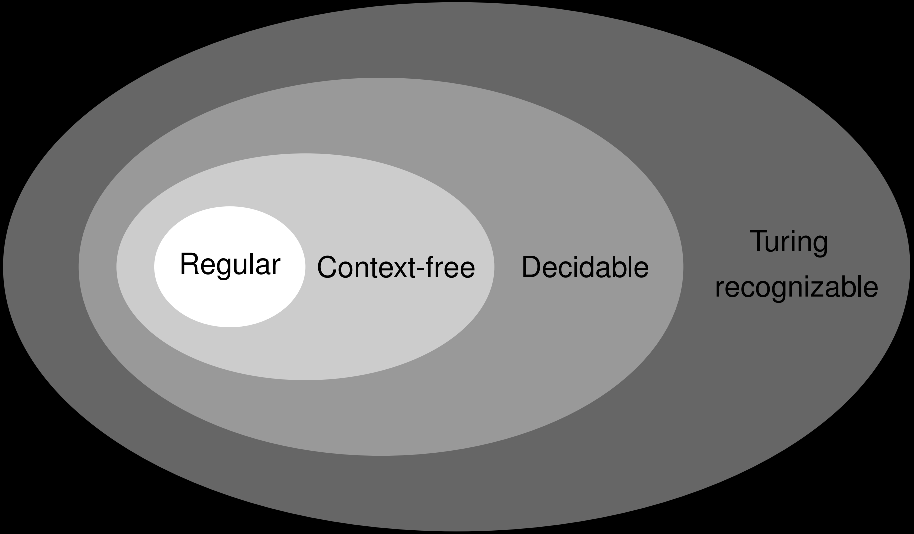
Dr Kamal Bentahar

School of Engineering, Environment and Computing
Coventry University

21/11/2014


380CT
Complexity
NP
P
NP-hard
NP-complete
Meta-heuristics
Decidability
Reduction
Turing Unrecognizable
"Halting problem"
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM

The "computation universe" discovered so far...



Time Complexity

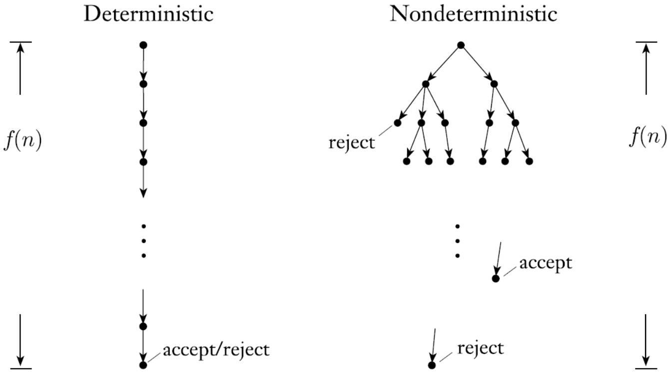
- Being **decidable** means that an **algorithm** exists to decide the problem.
- However, the algorithm may still be *practically* ineffective because of its **time** and/or **space** cost.

Running time / Time complexity

The **running time** or **time complexity** of a TM that always halts is the <u>maximum</u> number of steps $f(n)$ that it makes on any input of length n.

For nondeterministic TMs consider **all the branches** of its computation.

We say that it runs in time $f(n)$; and that it is an $f(n)$-time TM.



Big-O notation cheat-sheet

1. 1 — constant, does not depend on n
2. $\log n$ — think of this as n^ε for a “small” ε
3. n
4. $n \log n$ — think $n \times n^\varepsilon = n^{1+\varepsilon}$
5. n^2
6. $n^2 \log n$ — think $n^{2+\varepsilon}$
7. n^3
8. 2^n
9. 3^n
10. $n!$
11. n^n

Big-O notation cheat-sheet

- Constant $O(1)$
- Polynomial $O(n), O(n^2), \ldots, O(n^k), \ldots$ $(k \geq 1)$
- Exponential $O(2^n), O(3^n), \ldots, O(b^n), \ldots$ $(b > 1)$
- Factorial $O(n!)$
- $O(n^n)$

"Tricks":

$$n^k \log n \quad \sim \quad n^k \, n^\varepsilon = n^{k+\varepsilon}$$

$$n! \quad \sim \quad n^n / e^n \quad \text{where } e = 2.718\ldots$$

2013	$O(1)$
$n + 5$	$O(n)$
$543n + n^3 + 13$	$O(n^3)$
$4n^2 + 2784n + 10^{74}$	$O(n^2)$
$n^{578} + 4685 + 2^n$	$O(2^n)$
$n + n \log n + 35$	$O(n \log n)$
$n^2 + n \log n + 35$	$O(n^2)$
$n^2 + n^2 \log n + 35$	$O(n^2 \log n)$
$n^{86754} + n!$	$O(n!)$

Formal definition of big-O notation

Big-O notation

Let f and g be functions

$$f, g\colon \mathbb{N} \to \mathbb{R}^+$$

Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq cg(n).$$

We say that $g(n)$ is an **(asymptotic) upper bound** for $f(n)$.

Time complexity class

Interesting observation:

Complexity relationships among TM variants

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multi-tape TM has an equivalent $O(t^2(n))$ time single-tape TM.

Time complexity class

Let $t\colon \mathbb{N} \to \mathbb{R}^+$ be a function.
Define the **time complexity class**

$$TIME(t(n)),$$

to be the collection of all languages that are decidable by an $O(t(n))$ time TM.

The class **P**

The class **P**

P is the class of languages that are decidable in polynomial time on a deterministic (single-tape) TM.

$$\mathbf{P} = \mathit{TIME}(1) \cup \mathit{TIME}(n) \cup \mathit{TIME}(n^2) \cup \mathit{TIME}(n^3) \cup \cdots .$$

P examples – path between two nodes in a graph

$PATH = \{\langle G, s, t\rangle \mid$ G is a directed graph that has a directed path from s to $t\}$.

A polynomial time algorithm for PATH

On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*.

P examples – relatively prime numbers

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$$

E : *Euclidean algorithm* for computing the *greatest common divisor*

On input $\langle x, y\rangle$, where x and y are natural numbers in binary:

1. Repeat until $y = 0$:
2. $\quad x \leftarrow x \mod y$
3. $\quad$ Exchange x and y.
4. Output x.

Algorithm that solves RELPRIME, using E as a subroutine:

On input $\langle x, y\rangle$, where x and y are natural numbers in binary:

1. Run E on $\langle x, y\rangle$.
2. If the result is 1, *accept*. Otherwise, *reject*."

The class **NP** – Solution vs Verification

- Solving: **finding/searching** for a solution.
- Verifying: **confirming** that a proposed solution is correct.

For example, given a candidate example of a tour around England, we just need to check if it

- contains all the required cities
- uses no city more than once
- finishes at its starting point
- uses only valid routes

The class **NP**

Verifiers

A verifier for a language L is an algorithm V, where

$$L = \{w \mid \ V \text{ accepts } \langle w, c\rangle \text{ for some } \textit{certificate} \text{ string } c\}.$$

- A **polynomial time verifier** runs in polynomial time in the length of w.
- A language is **polynomially verifiable** if it has a polynomial time verifier.

The class **NP**

NP is the class of languages that have polynomial time verifiers.

The class **NP**

Nondeterministic Polynomial time complexity class

$$NTIME(t(n)) = \{\text{Language decided by an } O(t(n)) \text{ time nondeterministic TM}\}.$$

$$\mathbf{NP} = NTIME(1) \cup NTIME(n) \cup NTIME(n^2) \cup NTIME(n^3) \cup \cdots .$$

NP examples – the subset sum problem

$$\begin{aligned} \mathit{SUBSETSUM} &= \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \\ &\quad \text{and for some } \{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\} \colon y_1 + \cdots + y_\ell = t\}. \end{aligned}$$

Verifier: "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, *accept*. Otherwise, *reject*."

Alternatively, polynomial time NDTM: "On input $\langle S, t\rangle$:

1. Non-deterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, *accept*. Otherwise, *reject*."

NP examples – Hamiltonian path

Hamiltonian paths

A **Hamiltonian path** in a directed graph is a path that goes through each node exactly once.

$HAMPATH = \{\langle G, s, t\rangle \mid$ Directed graph G has a Hamiltonian path from s to $t\}$.

NP examples – cliques in a graph

Cliques

A **clique** in an undirected graph is a subgraph, wherein every two nodes are connected by an edge.
A k**-clique** is a clique that contains k nodes.

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}.$$

Verifier for $CLIQUE$: "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, *accept*. Otherwise, *reject*."

Alternatively, polynomial time NDTM: "On input $\langle G, k\rangle$, where G is a graph:

1. Nondeterministically select a subset c of k nodes of G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, accept; otherwise, reject."

Summary

P — Polynomial-time

Class of languages that are decidable in polynomial time.

$$\mathbf{P} = \bigcup_{k \geq 0} \mathit{TIME}(n^k).$$

NP — Nondeterministic Polynomial time

Class of languages that have polynomial time verifiers.

$$\mathbf{NP} = \bigcup_{k \geq 0} \mathit{NTIME}(n^k).$$

P = NP?