What hard problems mean?

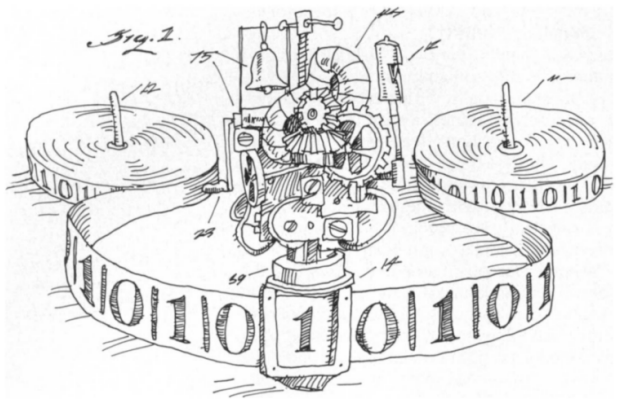
Arnaud Casteigts
University of Geneva

Science Faculty dinner

June 12, 2025.

(Partie I)

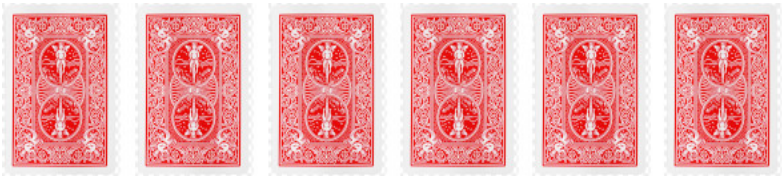
Fun with algorithms



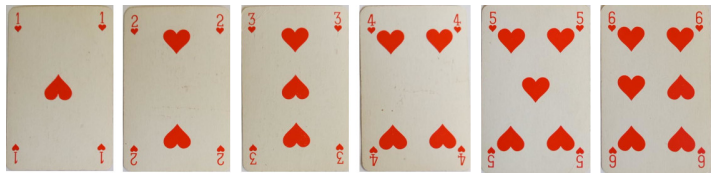
Sorting cards | playing dominos

Game 1: Sorting cards (without looking!)

The goal:



→



The rules: (1) Pick two cards, (2) Ask for comparison, (3) Rearrange the deck, (4) Repeat.

Game 2: Playing dominos

Input: a finite set of types of dominos (each available in infinite number)

E.g. $\frac{ca}{a}$ $\frac{a}{ab}$ $\frac{b}{ca}$ $\frac{abc}{c}$

Solution:

Question: Can you find a sequence of such dominos,
whose concatenation gives the same word **above** and **below** the line?

Which problem is the most difficult?

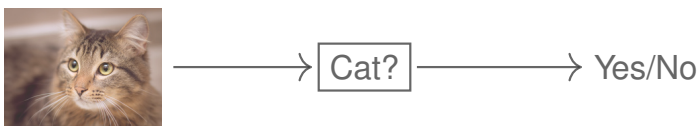
The framework

Examples of problems

Sorting:

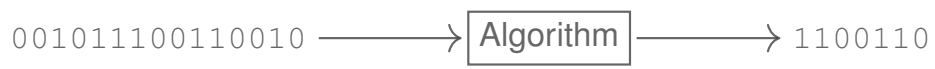
Itinerary:

Classification:



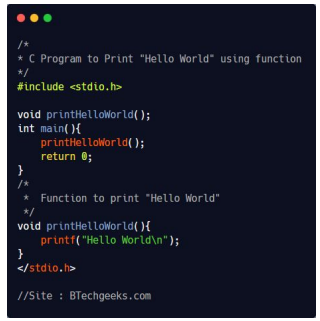
Mathematics:

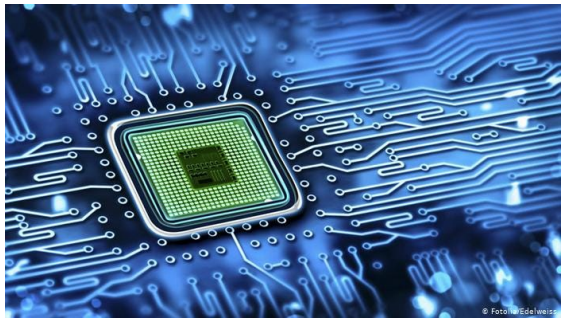
More generally



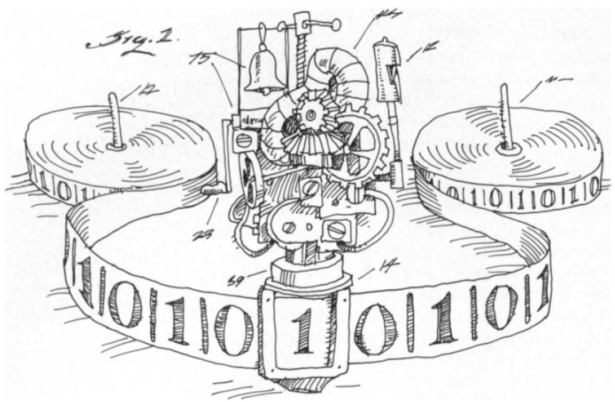
An algorithm takes as input a sequence of **symbols** and produce another one. It processes information.

What model of computation?





Turing Machine (1936)



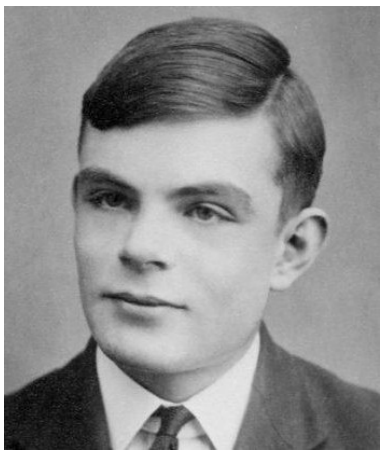
230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

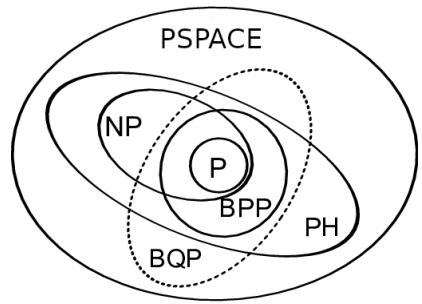
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*. it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are,



→ **Mathematical** model of an algorithm, presumably able to simulate any physically realizable algorithmic process (Church-Turing conjecture).

(Partie 2)

Computational complexity



Computational complexity?

Amount of **resources** required for solving a given problem.

What type of resources?

- Time (number of operations)
- Space (amount of memory)
- ...

Asymptotic point of view

- How does the cost scale as the input size n grows large?
- Notations $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ (ignore constant factors and dominated terms)

  Intuition $\leq \quad \geq \quad =$

  E.g.: $3n^2 + 5n + 4 = \Theta(n^2)$
- Some adjectives:

Constant	$\Theta(1)$	Exponential	$\Theta(2^n)$
Linear	$\Theta(n)$	Logarithmic	$\Theta(\log n)$
Quadratic	$\Theta(n^2)$	Quasi-linear	$\Theta(n \cdot \log n)$

Polynomial $O(n^c) = n^{O(1)}$

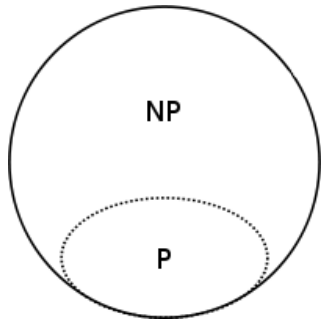
Complexity of a problem: How much does it cost as a function of n, **in the worst case**?

SORTING takes $O(n \cdot \log n)$ time. (polynomial ✓)

P versus NP

Famous complexity classes

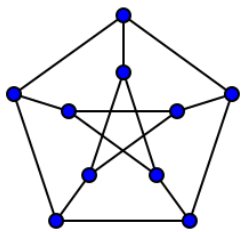
- Class P: Problems whose solution can be **found** in polynomial time.
- Class NP: Problems whose solution can be **verified** in polynomial time.
- PSPACE, EXP, BPP, BQP, IP, . . .

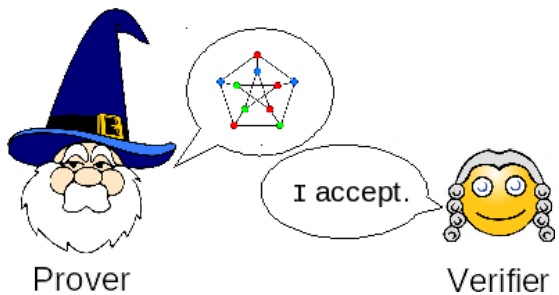


Example: 3-COLORING

Given a graph, can it be colored with 3 colors?

(neighbors must have $\neq$ colors)



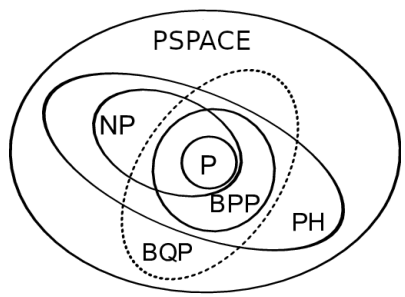


Best known algorithm is exponential!

Likely not in P... but clearly in NP ✓

Does P equal NP?

$\rightarrow$ One of the deepest questions in computer science
(one of the 7 millennium questions - $\$1\,M$)
$\rightarrow$ Impacts on philosophy, mathematics, logics
$\rightarrow$ Practical implications (everyday problems + cryptography)

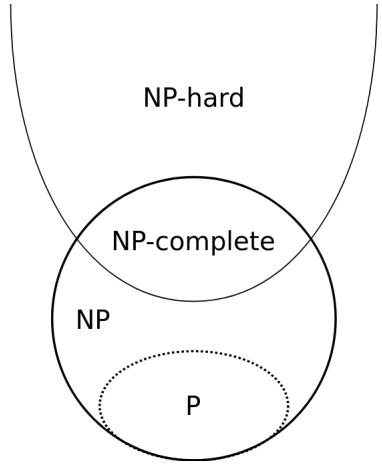


Many similar questions in the field, most of which are still open...

NP-completeness

Difficulty and completeness

- NP-hard: problems **as hard as** any other problem in NP
- NP-complete: both NP-hard and in NP.



If **any** NP-complete problem is solvable in polynomial time, they all are, and P $=$ NP.

If **any** NP-complete problem requires super-polynomial time, they all do, and P $\neq$ NP.

To keep in mind

This theory is concerned with **worst case** complexity. In practice, many NP-hard problems are tractable for real-world inputs, which are typically not **adversarial**.

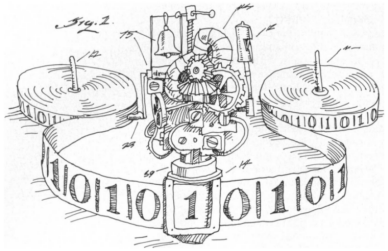
Examples of NP-complete problems

- SAT: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \vee \ldots$ (Cook, Levin'71)
- 3-Coloring, Clique, Set Cover, Hamiltonian Cycle, Tsp,
- Thousands of other problems ...

How to show that a problem is NP-hard?

$\rightarrow$ take another NP-hard problem, and reduce it to your problem.

What about AI?



V.S.

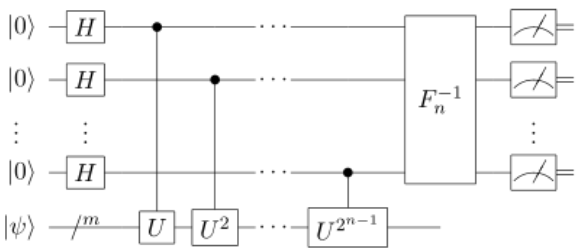
All the things we said apply to AI.

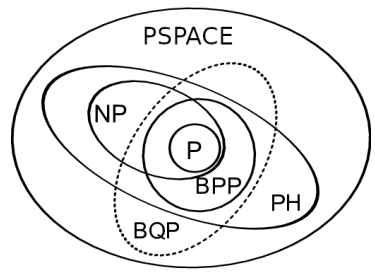
AI can work well in practice, because it targets **typical** inputs, not **adversarial** ones. It won't solve problems that are **hard on average**, like FACTORING.

What about quantum computers?

Class BQP: *Bounded error quantum polynomial time*

- Problems solvable in polynomial time by a **quantum computer**, with probability of success $> 1/2$.





Unlikely to solve NP-hard problems

However, some important problems like FACTORING are in BQP, with important consequences.

Back to dominos

In 1936, Turing showed that certain properties of programs are **undecidable**

→ No algorithm exists that can successully decide the property for **all** programs.

Example: “Given a program, is it going to terminate?” is an undecidable problem.

How about dominos?

ca / a	a / ab	b / ca	abc / c

It turns out every turing machine can be **encoded** into a set of dominos, in such a way that:

A solution for these dominos exists if and only if the machine terminates.

→ The domino game has to be **undecidable** too!

Et toc :-)