Basics of computational complexity

G. Ferrari Trecate

Dipartimento di Ingegneria Industriale e dell'Informazione
Università degli Studi di Pavia

Industrial Automation

Introduction

Problems: 1) quantify the efficiency of an algorithm
2) quantify the intrinsic difficulty of a problem

Instance of a problem

Ex. Problem (0): "sort positive integers $c_1, \dots c_n$ in increasing order"

Instance I of (0): $n=3$, $c_1=2$, $c_2=100$, $c_3=7$

↳ instance = input arguments

An instance I has size $|I|$ equal to the number of bits needed to code it

Previous ex. $|I| = \lceil \log_2 n \rceil + \sum_{j=1}^{n} \lceil \log_2 c_j \rceil$ $\quad \lceil x \rceil$ = minimum integer $\geq x$

Def. Let $f, g : \mathbb{N} \to \mathbb{R}$. Then, $f(n) = O(g(n))$ if ↷ g "dominates" f

$$\exists\, n_0, c : \; f(n) \leq c\, g(n) \quad \forall n \geq n_0$$

Previous example : $|I| = O(n \log_2 L)$ $\quad L = \max\{c_1, \dots, c_n\}$

Rmk. $O(1)$ = bounded function

How the complexity of an algorithm scales with $|I|$?

Computation time: number of elementary operations (comparisons, sums, multiplications, etc.) necessary to solve a problem instance

Rmk. The definition is rigorous only if an underlying computation model is defined (e.g. Turing machine)
↳ Church-Turing "thesis": all "reasonable" computation models are equivalent

The computation time can vary between instances of the same size

Ex. Ordering with bubble-sort: $O(d)$ in the best case (→ instance size)
$O(d^2)$ in the worst case

Algorithm complexity: maximal computation time for solving all problem instances of size d

↳ worst-case concept but independent of specific instances and the computer used for solving the problem

Notation. $T(d)$ is the algorithm complexity and d is the instance size

Def. An algorithm is polynomial if $\exists k \in \mathbb{N}$ such that $T(d) = O(d^k)$

Ex.
- Bubble-sort is polynomial $(T(d) = O(d^2))$
- Solving $Ax = b$, $A \in \mathbb{R}^{d \times d}$ with Gauss elimination requires $T(d) = O(d^3)$ → polynomial
- Simplex (n unknowns, m constraints)

$$T(n,m) = O\left(\frac{n!}{m!(n-m)!}\right) \rightarrow \text{NOT polynomial}$$

Easy problems

Def. A problem is polynomial (or easy) if there is a polynomial algorithm for solving it. Otherwise the problem is hard.

Rmk. This is not a mathematical definition since it refers to the state of art.

Ex. LP problems have been hard up to 1979 (there was only the simplex algorithm)

1979: Khachiyan proposes the ellipsoid method that is polynomial (but less efficient than simplex, on average)

1984: Interior point methods

Complexity of hard problems

Def. A decision problem is a problem with yes/no answer.

Ex. (D1) Is there any $x \geq 0$ verifying $Ax \leq b$ $(A \in \mathbb{N}^{m \times n}, b \in \mathbb{N}^m)$?

(D2) Is the digraph $G = (V, E)$ connected?

(D3) Does the digraph $G = (V, E)$ contain a Hamiltonian cycle?

- (D1) and (D2) are polynomial → for (D2) we will discuss a polynomial algorithm in the next lectures
- (D1), (D2) and (D3) have the following property: it is possible to certify a "yes" answer in polynomial time using the problem solution

Ex. Problem (D1). The algorithm computes x^* verifying

$$Ax^* \leq b$$

Certification: test $a_{j,\cdot}\, x^* \leq b_j$, $j = 1, \dots, m$ and $x_i^* \geq 0$, $i = 1, \dots n$
(nm products and n+m comparisons)
↳ polynomial time

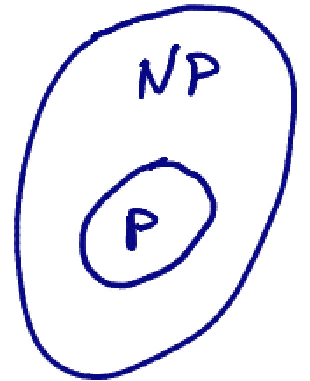
Problem (D3). The algorithm computes a cycle c

Certification: verify that c is an Hamiltonian cycle
↳ polynomial in the number of vertices

NP problems

Def. A polynomial certificate is an auxiliary piece of information that allows one to certify the correctness of an answer in polynomial time

Def. A decision problem belongs to the class NP (Nondeterministic Polynomial) if every instance with "yes" answer has a polynomial certificate



Rmk. Certifying "yes" answers of a polynomial problem can be done in polynomial time by solving the problem itself

↳ $P \subseteq NP$ (P = class of easy problems)

Ex. Problem (D3) is NP

How to compare NP problems?

Def. A decision problem P_1 can be reduced to P_2 in polynomial time (notation $P_1 \propto P_2$) if there is an algorithm for solving P_1 that

1) calls a polynomial number of times an algorithm for solving P_2
2) if P_2 was $O(1)$ the whole algorithm is polinomial

Rmk. Often $P_1 \propto P_2$ is shown by proving that every instance of P_1 with yes/no answer can be transformed in polynomial time into an instance of P_2 with the same answer

↳ the algorithm for solving P_2 is called just once. The polynomial complexity of data transformation is covered in point (2) above.

Example

Given a digraph $G = (V, E)$

P_1 : is the digraph connected?

P_2 : given two vertices v_1 and v_2, is there a path from v_1 to v_2?

↳ $P_1 \propto P_2$

- Call P_2 n^2 times ($n = |V|$), i.e for all pairs (v_1, v_2)

Rmk. •) $\propto$ is a transitive relation, i.e. $P_1 \propto P_2$, $P_2 \propto P_3 \Rightarrow P_1 \propto P_3$

•) Useful to study the complexity of a problem by reducing it to a problem with known complexity

Def. $P_2 \in NP$ is *NP-complete* if $\forall P_1 \in NP$ one has $P_1 \propto P_2$

Ex. Problem (D3) is NP-complete (the proof is difficult).

Lemma. If $P_1 \in NP$, P_2 is NP-complete and $P_2 \propto P_1$, then P_1 is NP-complete $\rightarrow$ useful for showing an NP problem is NP-complete!

Proof. P_2 NP-complete $\Rightarrow P \propto P_2$, $\forall P \in NP$. Since $P_2 \propto P_1$, then $P \propto P_2 \propto P_1$ and hence $P \propto P_1$

NP-hard problems

Def. A problem P^*, not necessarily in NP, is NP-hard if

$$\forall P_1 \in NP,\ P_1 \propto P^*$$

Rmk. P^* is at least as difficult as every NP problem

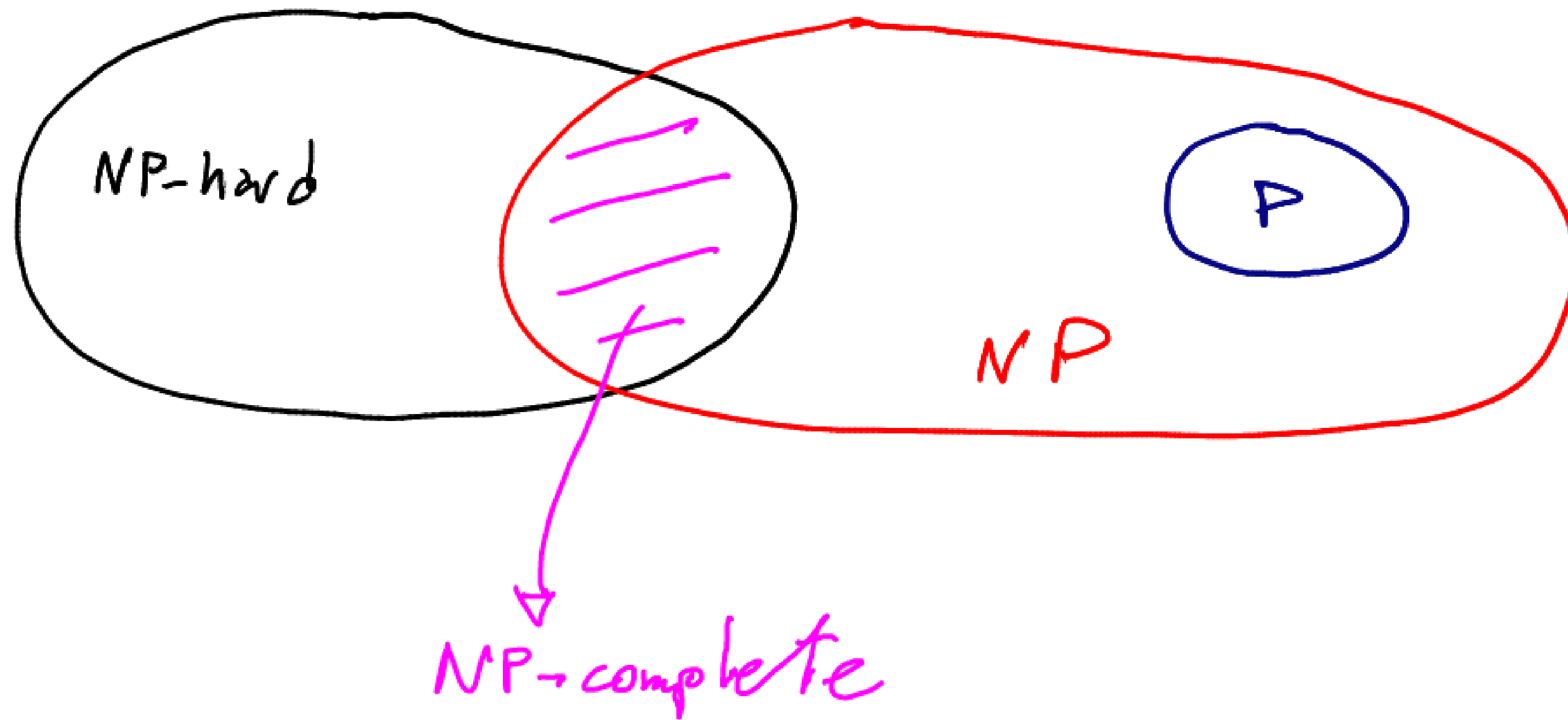
If one finds a polynomial algorithm for solving a single NP-complete (or NP-hard) problem, one would have

$$P = NP$$

↳ a Millenium Prize problem !

↳ most scientists expect $P \neq NP$

Relations among complexity classes



Complexity of problems seen in the previous lecture

Problem A: Is an undirected graph connected?

Problem B: Given a digraph and two nodes v_1 and v_2, check if v_2 is connected to v_1.

↳ Both A and B are polynomial (algorithms in the next lectures)

Problem C: Does a directed graph contain a Hamiltonian cycle?

↳ C is NP-complete

TSP: given an undirected complete network and $z \in \mathbb{R}$, check if there is a Hamiltonian cycle of cost less than z

↳ TSP is NP-complete

TSP in optimization form: as in TSP but compute a Hamiltonian cycle of minimal cost

↳ NP-hard

In general, "optimization forms" of NP-complete problems are NP-hard