Chapter 8
Complexity

8.1 Introduction

- Solvable problems versus efficiently solvable problems.
- Measuring complexity: complexity functions.
- Polynomial complexity.
- NP-complete problems.

8.2 Measuring complexity

- Abstraction with respect to the machine being used.
- Abstraction with respect to the data (data size as only parameter).
- O notation.
- Efficiency criterion: polynomial.

8.3 Polynomial problems

- Influence of the encoding.
- Graph example.
- Reasonable encodings:
  - no padding,
  - polynomial decoding,
  - unary representation of numbers not allowed.

Complexity and Turing machines

Time complexity of a Turing machine that always stops:

$$T_M(n) = \max \{m \mid \exists x \in \Sigma^*, |x| = n \text{ and the execution of } M \text{ on } x \text{ is } m \text{ steps long}\}.$$

A Turing machine is polynomial if there exists a polynomial $p(n)$ such that

$$T_M(n) \leq p(n)$$

for all $n \geq 0$.

The class P is the class of languages that are decided by a polynomial Turing machine.

8.4 Polynomial transformations

- Diagonalisation is not adequate to prove that problems are not in P.

- Another approach: comparing problems.

The Travelling Salesman (TS)

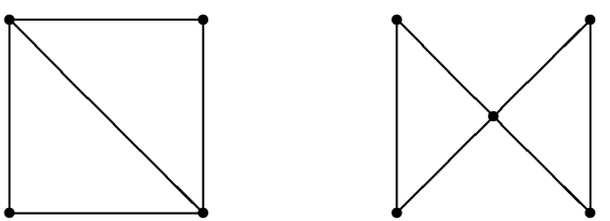
- Set C of n Cities.
- Distances $d(c_i, c_j)$.
- A constant b.
- Is there a permutation of the towns such that:

$$\sum_{1 \leq i < n} d(c_{p_i}, c_{p_{i+1}}) + d(c_{p_n}, c_{p_1}) \leq b.$$

Hamiltonian Circuit (HC)

- Graph $G = (V, E)$

- Is there a closed circuit in the graph that contains each vertex exactly once.



Definition of polynomial transformations

Goal : to establish a link between problems such as HC and TS (one is in P if and only if the other is also in P).

Definition :
Consider languages $L_1 \in \Sigma_1^*$ and $L_2 \in \Sigma_2^*$. A *polynomial transformation* from L_1 to L_2 (notation $L_1 \propto L_2$) is a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following conditions :

1. it is computable in polynomial time,

2. $f(x) \in L_2$ if and only if $x \in L_1$.

HC $\propto$ TS

- The set of cities is identical to the set of vertices of the graph, i.e. $C = V$.

- The distances are the following $(c_i, c_j) = \begin{cases} 1 & \text{si } (c_i, c_j) \in E \\ 2 & \text{si } (c_i, c_j) \notin E \end{cases}$.

- The constant b is equal to the number of cities, i.e. $b = |V|$.

Properties of $\propto$

If $L_1 \propto L_2$, then

- if $L_2 \in \mathsf{P}$ then $L_1 \in \mathsf{P}$,
- if $L_1 \notin \mathsf{P}$ then $L_2 \notin \mathsf{P}$.

If $L_1 \propto L_2$ et $L_2 \propto L_3$, then

- $L_1 \propto L_3$.

Polynomially equivalent problems

Definition
Two languages L_1 and L_2 are *polynomially equivalent* (notation $L_1 \equiv_P L_2$) if and only if $L_1 \propto L_2$ and $L_2 \propto L_1$.

- Classes of polynomially equivalent problems: either all problems in the class are in P, or none is.
- Such an equivalence class can be built incrementally by adding problems to a known class.
- We need a more abstract definition of the class containing HC and TS.

The class NP

- The goal is to characterise problems for which it is necessary to examine a very large number of possibilities, but such that checking each possibility can be done quickly.

- Thus, the solution is fast, if enumerating the possibilities does not cost anything.

- Modelisation : nondeterminism.

The complexity of nondeterministic Turing machines

The execution time of a nondeterministic Turing machine on a word w is given by

- the length of the *shortest* execution accepting the word, if it is accepted,
- the value 1 if the word is not accepted.

The time complexity of M (non deterministic) is the function $T_M(n)$ defined by

$$T_M(n) = \max \{m \mid \exists x \in \Sigma^*, |x| = n \text{ and the execution time of } M \text{ on } x \text{ is } m \text{ steps long}\}.$$

The definition of NP

Définition

The class NP (from *N*ondeterministic *P*olynomial) is the class of languages that are accepted by a polynomial nondeterministic Turing machine.

Exemple

HC and TS are in NP.

Theorem

Consider $L \in$ NP. There exists a deterministic Turing machine M and a polynomial $p(n)$ such that M decides L and has a time complexity bounded by $2^{p(n)}$.

Let M_{nd} be a nondeterministic machine of polynomial complexity $q(n)$ that accepts L. The idea is to simulate all executions of M_{nd} of length less than $q(n)$. For a word w, the machine M must thus:

1. Determine the length n of w and compute $q(n)$.

2. Simulate each execution of M_{nd} of length $q(n)$ (let the time needed be $q'(n)$). If r is the largest number of possible choices within an execution of M_{nd}, there are at most $r^{q(n)}$ executions of length $q(n)$.

3. If one of the simulated executions accepts, M accepts. Otherwise, M stops and rejects the word w.

Complexity : bounded by $r^{q(n)} \times q'(n)$ and thus by $2^{\log_2(r)(q(n)+q'(n))}$, which is of the form $2^{p(n)}$.

The structure of NP

Definition A polynomial equivalence class C_1 is smaller than a polynomial equivalence class C_2 (notation $C_1 \preceq C_2$) if there exists a polynomial transformation from every language in C_1 to every language in C_2.

Smallest class in NP : P

- The class NP contains the class P (P $\subseteq$ NP).
- The class P is a polynomial equivalence class.
- For every $L_1 \in$ P and for every $L_2 \in$ NP, we have $L_1 \propto L_2$.

Largest class in NP : NPC

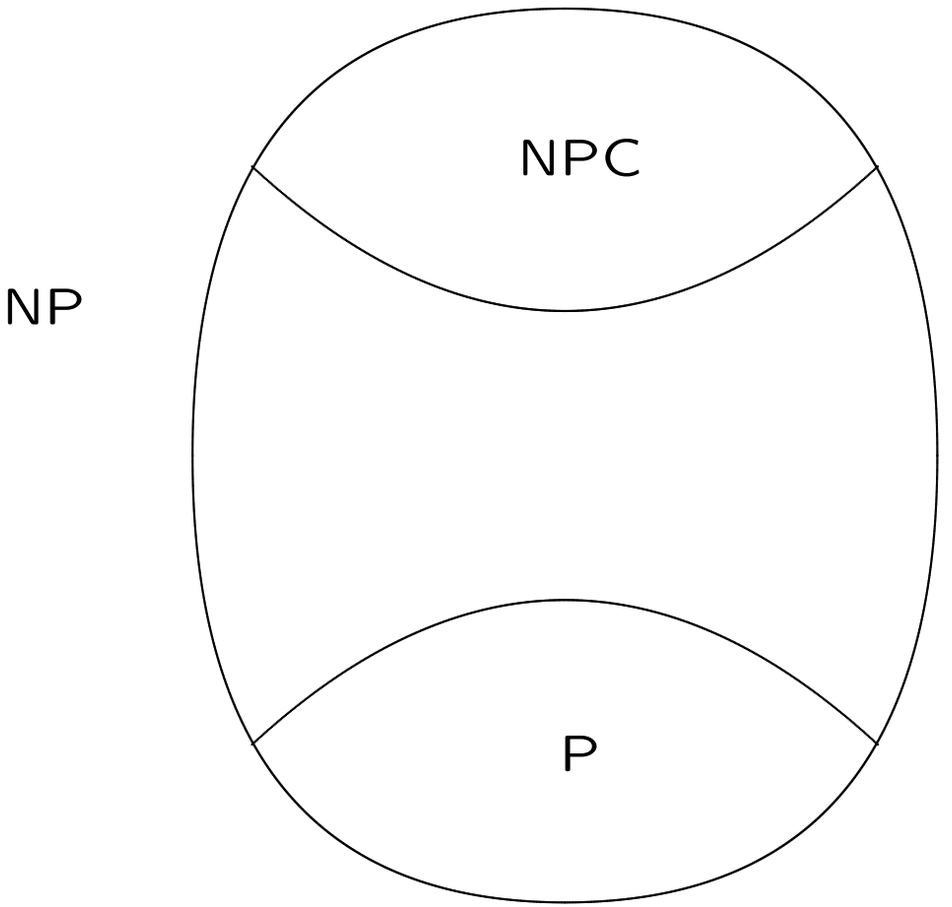
A language L is NP-complete if

1. $L \in$ NP,

2. for every language $L' \in$ NP, $L' \propto L$.

Theorem
If there exists an NP-complete language L decided by a polynomial algorithm, then all languages in NP are polynomially decidable, i.e. P = NP.

Conclusion : An NP-complete problem does not have a polynomial solution if and only if P $\neq$ NP


NP
NPC
P

Proving NP-completeness

To prove that a language L is NP-complete, one must establish that

1. it is indeed in the class NP ($L \in$ NP),

2. for every language $L' \in$ NP, $L' \propto L$,

or, alternatively,

3. There exists $L' \in$ NPC such that $L' \propto L$.

Concept of NP-hard problem.

A first NP-complete problem
propositional calculus

Boolean calculus :

p	$\neg p$
0	1
1	0

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

p	q	$p \supset q$
0	0	1
0	1	1
1	0	0
1	1	1

- Boolean expression: $(1 \wedge (0 \vee (\neg 1))) \supset 0$.

- Propositional variables and propositional calculus :
  $(p \wedge (q \vee (\neg r))) \supset s$.

- Interpretation function. Valid formula, satisfiable formula.

- Conjunctive normal form: conjunction of disjunctions of literals.

Cook's theorem

SAT Problem : satisfiability of conjunctive normal form propositional calculus formulas.

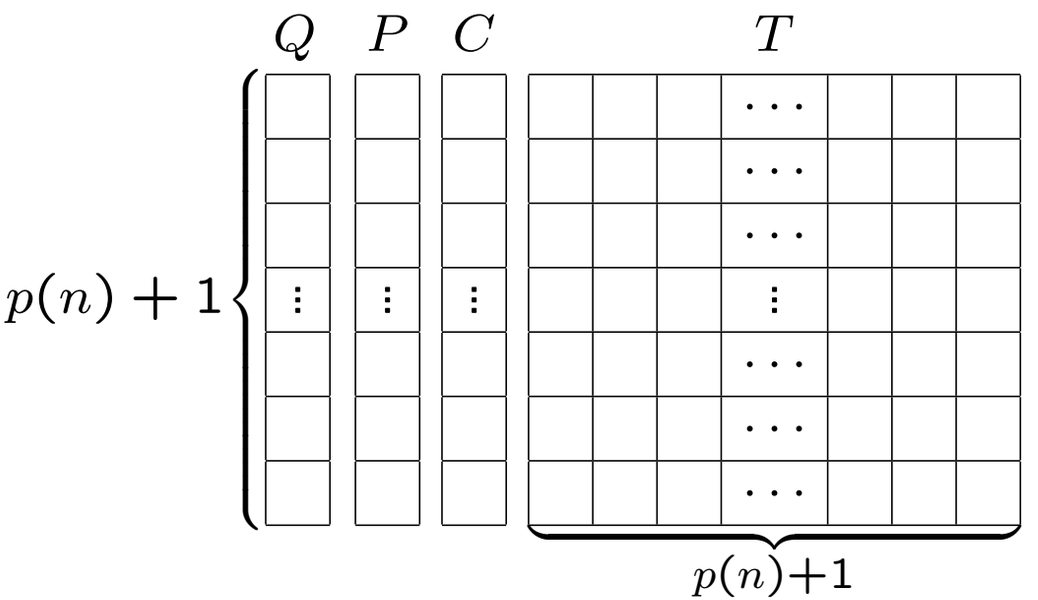
Theorem
The SAT problem is NP-complete

Proof

1. SAT is in NP.
2. There exists a polynomial transformation from every language in NP to L_{SAT}.
   - Transformation with two arguments : word and language.
   - The languages of NP are characterised by a polynomial-time nondeterministic Turing machine.

Word w ($|w| = n$) and nondeterministic polynomial Turing machine $M = (Q, \Gamma, \Sigma, \Delta, s, B, F)$ (bound $p(n)$).

Description of an execution of M (T : tape; Q : state; P : position; C : choice.)



Representing an execution with propositional variables:

1. A proposition $t_{ij\alpha}$ for $0 \leq i, j \leq p(n)$ and $\alpha \in \Gamma$.

2. A proposition $q_{i\kappa}$ for $0 \leq i \leq p(n)$ and $\kappa \in Q$.

3. A proposition p_{ij} for $0 \leq i, j \leq p(n)$.

4. A proposition c_{ik} for $0 \leq i \leq p(n)$ and $1 \leq k \leq r$.

Formula satisfied only by an execution of M that accepts the word w : conjunction of the following formulas.

$$\bigwedge_{0 \leq i,j \leq p(n)} \left[\left(\bigvee_{\alpha \in \Gamma} t_{ij\alpha} \right) \wedge \bigwedge_{\alpha \neq \alpha' \in \Gamma} (\neg t_{ij\alpha} \vee \neg t_{ij\alpha'}) \right]$$

One proposition for each tape cell. Length $O(p(n)^2)$.

$$\bigwedge_{0 \leq i \leq p(n)} \left[\left(\bigvee_{0 \leq j \leq p(n)} p_{ij} \right) \wedge \bigwedge_{0 \leq j \neq j' \leq p(n)} (\neg p_{ij} \vee \neg p_{ij'}) \right]$$

One proposition for each position. Length $O(p(n)^3)$.

$$\left[\bigwedge_{0\leq j\leq n-1} t_{0jw_{j+1}} \wedge \bigwedge_{n\leq j\leq p(n)} t_{0jB}\right] \wedge q_{0s} \wedge p_{00}$$

Initial state. Length $O(p(n))$

$$\bigwedge_{\substack{0\leq i<p(n)\\ 0\leq j\leq p(n)\\ \alpha\in\Gamma}} [(t_{ij\alpha} \wedge \neg p_{ij}) \supset t_{(i+1)j\alpha}]$$

$$\bigwedge_{\substack{0\leq i<p(n)\\ 0\leq j\leq p(n)\\ \alpha\in\Gamma}} [\neg t_{ij\alpha} \vee pij \vee t_{(i+1)j\alpha}]$$

Transitions, tape not modified. Length $O(p(n)^2)$.

$$\bigwedge_{\substack{0\le i<p(n)\\ 0\le j\le p(n)\\ \alpha\in\Gamma\\ 1\le k\le r}} \left[\begin{array}{l} ((q_{i\kappa}\wedge p_{ij}\wedge t_{ij\alpha}\wedge c_{ik})\supset q_{(i+1)\kappa'})\wedge \\ ((q_{i\kappa}\wedge p_{ij}\wedge t_{ij\alpha}\wedge c_{ik})\supset t_{(i+1)j\alpha'})\wedge \\ ((q_{i\kappa}\wedge p_{ij}\wedge t_{ij\alpha}\wedge c_{ik})\supset p_{(i+1)(j+d)}) \end{array}\right]$$

$$\bigwedge_{\substack{0\le i<p(n)\\ 0\le j\le p(n)\\ \alpha\in\Gamma\\ 1\le k\le r}} \left[\begin{array}{l} (\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg t_{ij\alpha}\vee\neg c_{ik}\vee q_{(i+1)\kappa'})\wedge \\ (\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg t_{ij\alpha}\vee\neg c_{ik}\vee t_{(i+1)j\alpha'})\wedge \\ (\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg t_{ij\alpha}\vee\neg c_{ik}\vee p_{(i+1)(j+d)}) \end{array}\right]$$

Transitions, modified part. Length $O(p(n)^2)$.

$$\bigvee_{\substack{0 \leq i \leq p(n) \\ \kappa \in F}} [q_{i\kappa}]$$

Final state reached. Length $O(p(n))$.

- Total length of the formula $O(p(n)^3)$.
- The formula can be built in polynomial time.
- Thus, we have a transformation that is polynomial in terms of $n = |w|$.
- The formula is satisfiable if and only if the Turing machine M accepts.

Other NP-complete problems

3-SAT : satisfiability for conjunctive normal form formulas with exactly 3 literals per clause.

SAT $\propto$ 3-SAT.

1. A clause $(x_1 \vee x_2)$ with two literals is replaced by

$$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$

2. A clause (x_1) with a single literal is replaced by

$$\begin{array}{ccccc} (x_1 \vee y_1 \vee y_2) & \wedge & (x_1 \vee y_1 \vee \neg y_2) & \wedge \\ (x_1 \vee \neg y_1 \vee y_2) & \wedge & (x_1 \vee \neg y_1 \vee \neg y_2) & \end{array}$$

3. A clause

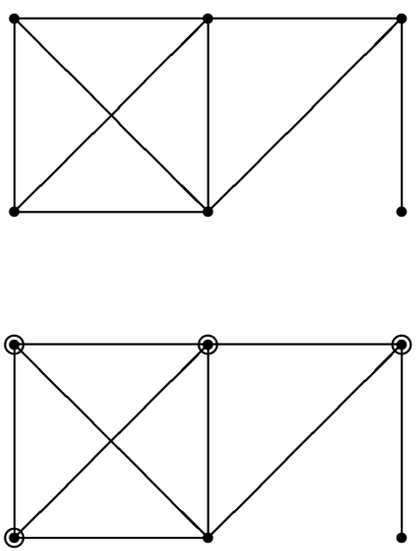
$$(x_1 \vee x_2 \vee \cdots \vee x_i \vee \cdots \vee x_{\ell-1} \vee x_\ell)$$

with $\ell \geq 4$ literals is replaced by

$$\begin{aligned}(x_1 \vee x_2 \vee y_1) \;& \wedge (\neg y_1 \vee x_3 \vee y_2) \\ & \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \cdots \\ & \wedge (\neg y_{i-2} \vee x_i \vee y_{i-1}) \wedge \cdots \\ & \wedge (\neg y_{\ell-4} \vee x_{\ell-2} \vee y_{\ell-3}) \\ & \wedge (\neg y_{\ell-3} \vee x_{\ell-1} \vee x_\ell)\end{aligned}$$

The *vertex cover* problem (VC) is NP-complete.

Given a graph $G = (V, E)$ and an integer $j \leq |V|$, the problem is to determine if there exists a subset $V' \subseteq V$ such that $|V'| \leq j$ and such that, for each edge $(u, v) \in E$, either u, or $v \in V'$.



3-SAT $\propto$ VC

Instance of 3-SAT :

$$E_1 \wedge \cdots \wedge E_i \wedge \cdots \wedge E_k$$

Each E_i is of the form

$$x_{i1} \vee x_{i2} \vee x_{i3}$$

where x_{ij} is a literal. The set of propositional variables is

$$\mathcal{P} = \{p_1, \ldots, p_\ell\}.$$

The instance of VC that is built is then the following.

1. The set of vertices V contains

   (a) a pair of vertices labeled p_i and $\neg p_i$ for each propositional variable in $\mathcal{P}$,

   (b) a 3-tuple of vertices labeled x_{i1}, x_{i2}, x_{i3} for each clause E_i.

   The number of vertices is thus equal to $2\ell + 3k$.

2. The set of edges E contains

   (a) The edge $(p_i, \neg p_i)$ for each pair of vertices $p_i, \neg p_i$, $1 \leq i \leq \ell$,

   (b) The edges (x_{i1}, x_{i2}), (x_{i2}, x_{i3}) et (x_{i3}, x_{i1}) for each 3-tuple of vertices x_{i1}, x_{i2}, x_{i3}, $1 \leq i \leq k$,

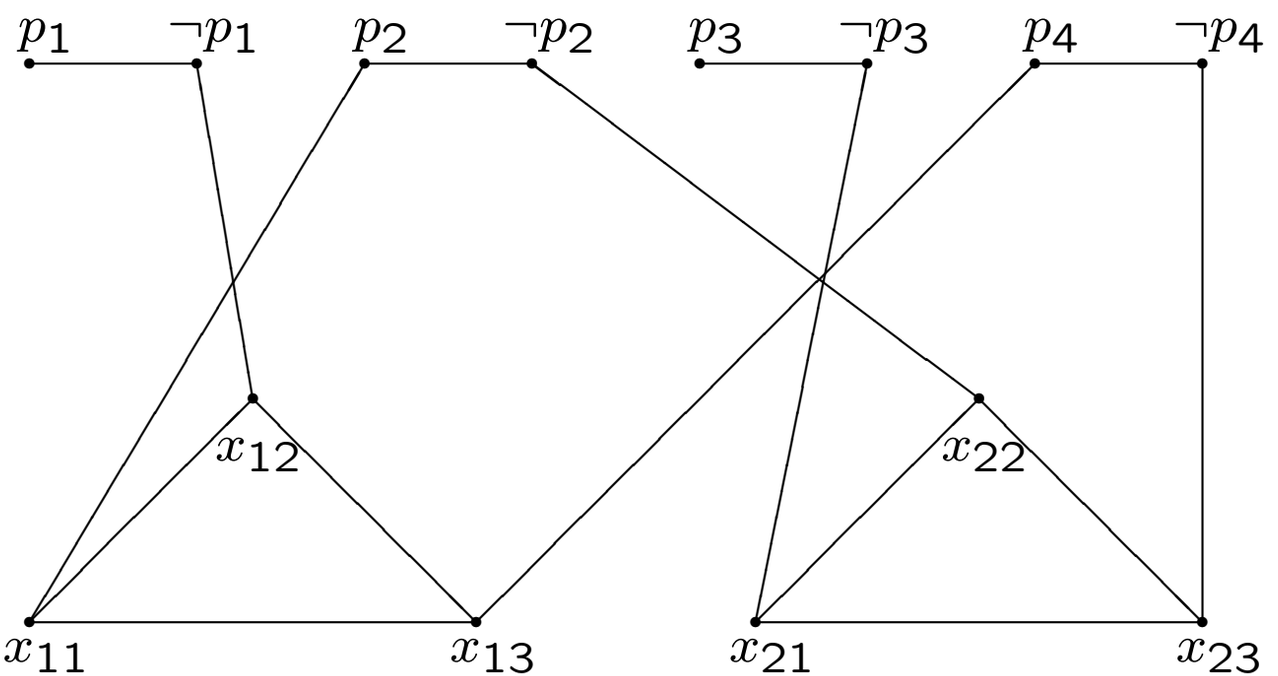
   (c) an edge between each vertex x_{ij} and the vertex p or $\neg p$ representing the corresponding literal.

   The number of edges is thus $\ell + 6k$.

3. The constant j is $\ell + 2k$.

Example

$$(p_2 \vee \neg p_1 \vee p_4) \wedge (\neg p_3 \vee \neg p_2 \vee \neg p_4)$$



Other examples

The Hamiltonian circuit (HC) and travelling salesman (TS) problems are NP-complete.

The *chromatic number* problem is NP-Complete. Given a graph G and a constant k this problem is to decide whether it is possible to colour the vertices of the graph with k colours in such a way that each pair of adjacent (edge connected) vertices are coloured differently.

The *integer programming* problem is NP-compete. An instance of this problem consists of

1. a set of m pairs $(\overline{v_i}, d_i)$ in which each $\overline{v_i}$ is a vector of integers of size n and each d_i is an integer,
2. a vector $\overline{d}$ of size n,
3. a constant b.

The problem is to determine if there exists an integer vector $\overline{x}$ of size n such that $\overline{x} \cdot \overline{v_i} \leq d_i$ for $1 \leq i \leq m$ and such that $\overline{x} \cdot \overline{d} \geq b$.

Over the rationals this problem can be solved in polynomial time (linear programming).

The problem of checking the equivalence of nondeterministic finite automata is NP-hard. Notice that there is no known NP algorithm for solving this problem. It is complete in the class PSPACE.

8.8 Interpreting NP-completeness

- Worst case analysis. Algorithms that are efficient "on average" are possible.

- Heuristic methods to limit the exponential number of cases that need to be examined.

- Approximate solutions for optimisation problems.

- The "usual" instances of problems can satisfy constraints that reduce to polynomial the complexity of the problem that actually has to be solved.

8.9 Other complexity classes

The class co-NP is the class of languages L whose complement ($\Sigma^* - L$) is in NP.

The class EXPTIME is the class of languages decided by a deterministic Turing machine whose complexity function is bounded by an exponential function ($2^{p(n)}$ where $p(n)$ is a polynomial).

The class PSPACE is the class of languages decided by a deterministic Turing machine whose space complexity (the number of tape cells used) is bounded by a polynomial.

The class NPSPACE is the class of languages accepted by a nondeterministic Turing machine whose space complexity is bounded by a polynomial.

$$\mathsf{P} \subseteq \begin{matrix}\mathsf{NP} \\ \text{co-NP}\end{matrix} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}.$$