# Complexité : classes P NP coNP EXP

M1 Informatique Luminy 2024-25


| | |
|---|---|
| Kévin Perrot | <kevin.perrot@univ-amu.fr> |
| Pablo Concha-Vega | <pablo.concha-vega@lis-lab.fr> |
| Antonio E. Porreca | <antonio.porreca@univ-amu.fr> |
| Mathieu Roget | <mathieu.roget@univ-amu.fr> |


9hCM 9hTD 9hTP

NF=MAX(ET ; 0.3*CC+0.7*ET)

# Différents types de problèmes

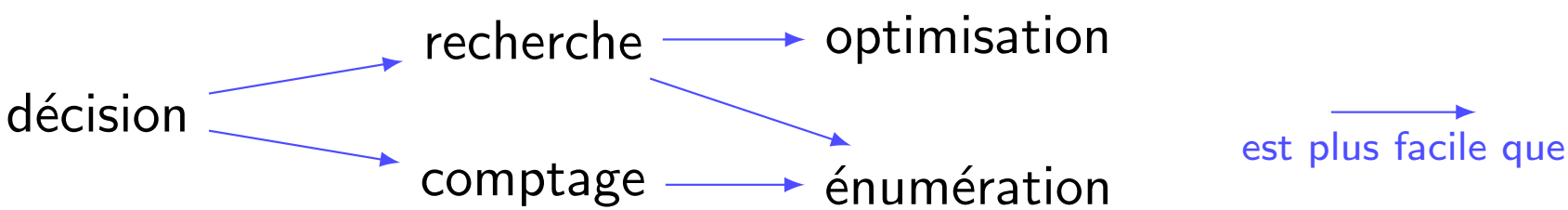
Étant donnée une entrée X. Problème de :

- décision : existe t-il un(e) Y ? est-ce que X a la propriété Z ?
- recherche : donner un(e) Y (si possible).
- comptage : combien existe-t-il de Y ?
- énumération : lister tous les Y (possiblement aucun).
- optimisation : donner un(e) Y qui maximise/minimise q(Y).

Exemples à décliner pour chaque type :

**SAT** (Satisfiabilité)
Entrée : une FNC $\varphi$ avec $n$ variables et $m$ clauses.
Question : $\varphi$ est-elle satisfaisable ?

**Clique**
Entrée : un graphe $G = (V, E)$ et un entier $k$.
Question : $G$ possède t-il une clique (sous-graphe complet) de taille $k$ ?




**Remarque.** Calcul de la valeur optimale par recherche dichotomique avec pb décision.

# Problèmes de décision

Dans la « vraie vie » les problèmes de décision sont peu intéressants.

Intérêts sur le plan **théorique** :

- bornes inférieures de complexité pour les autres types
- objets mathématiques les plus simples :
  - problème = langage
  - instance = mot

Il existe une branche de la théorie de la complexité pour chaque type de problème, et la branche principale concerne les problèmes de décision.

Pour passer d'un problème de décision à un langage, il faut un codage.

Le codage est souvent implicite, car standard : entiers, graphes, tableaux...

Chaque instance (entrée possible) est un mot,
et le problème de décision est le langage formé de
l'ensemble des instances positives (réponse « oui »).

Exemple : pour **SAT** au format DIMACS sur l'alphabet $\{0, 1, 2, \ldots, 9, -, \sqcup, \texttt{p}\sqcup\texttt{cnf}\}$ :

$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$

$\texttt{p}\sqcup\texttt{cnf}\sqcup\texttt{4}\sqcup\texttt{3}\sqcup\texttt{1}\sqcup\texttt{2}\sqcup\texttt{-3}\sqcup\texttt{0}\sqcup\texttt{-2}\sqcup\texttt{4}\sqcup\texttt{0}\sqcup\texttt{-1}\sqcup\texttt{3}\sqcup\texttt{-4}\sqcup\texttt{0} \in \mathbf{SAT}$

# Complexité algorithmique d'un problème (langage)

Un algorithme $A$ résoud (reconnait) un problème $L \subseteq \Sigma^*$ lorsque $A$ accepte $x \iff x \in L$

Complexité (en temps) dans le pire cas d'un problème (langage) $L \subseteq \Sigma^*$ :

$$T_L(n) = \inf_{\substack{\text{algo } A \\ \text{qui résoud } L}} \max_{w \in \Sigma^n} t_A(w)$$

↪ complexité en temps de l'algorithme le plus efficace (inf) dans le pire cas (max).

Intérêt de déterminer la complexité d'un problème ?

Comment déterminer la complexité d'un problème ?

- Borne supérieure de complexité : donner un algorithme. 😎
- Borne inférieure de complexité : montrer que aucun autre algorithme n'est plus efficace. 😰

**Peu** d'exemples démontrés de bornes inférieures de complexité.

- Trouver le minimum d'un tableau de $n$ éléments : $\Theta(n)$ optimal.
  ↪ car il faut bien consulter chacun des éléménts au moins une fois
- Trier un tableau de $n$ éléments : $\Theta(n \log n)$ comparaisons optimal.
  ↪ car l'arbre représentant les branchements possibles, avec pour noeuds internes les comparaisons, a pour feuilles les $n!$ permutations. $2^h \geq n! \Rightarrow h \geq \log(n!) \in \Theta(n \log n)$

Et surtout...

# Introduction du livre de Garey et Johson (1979)

...de **nombreux** exemples comme celui-ci :

# P versus NP

temps déterministe **P**olynomial  temps **N**on-déterministe **P**olynomial

 On ne parlera pas de non-déterminisme dans ce cours 

---

Les classes de complexité sont des ensembles de problèmes de décision (langages).

$\mathrm{DTIME}(T(n)) = \{L \mid L \text{ est reconnu par un programme } P \text{ en temps } \mathcal{O}(T(n))\}$
$\mathrm{DSPACE}(T(n)) = \{L \mid L \text{ est reconnu par un programme } P \text{ en espace } \mathcal{O}(T(n))\}$
↪ Déterministe

**Proposition.**
Pour toute fonction $t(n)$ on a $\mathrm{DTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$.
Pour toute fonction $s(n)$ on a $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(s(n))})$.

↪ arrêt et nombre de configurations utilisant au plus $s(n)$ bits (avec ligne du programme)

# Théorème de hiérarchie en temps

**Théorème de hiérarchie en temps.**
Soit $g(n)$ une fonction calculable en temps $\mathcal{O}(g(n))$, et
$f(n)$ asympt. négligeable devant $g(n)$ : telle que $f(n)\log f(n) \in o(g(n))$.
$\hookrightarrow$ MT
Alors $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$.

**Preuve.** Par diagonalisation. Soit $U$ un programme universel efficace, et :

$$D = \{\langle P\rangle \mid U(\langle P\rangle, \langle P\rangle) \text{ rejette en } \leq g(|\langle P\rangle|) \text{ étapes de } U\}$$

- $D \in \mathrm{DTIME}(g(n))$ : programme $G$ en $\mathcal{O}(g(n))$ qui, sur l'entrée $\langle P\rangle$ :
  1. calcule $g(|\langle P\rangle|)$
  2. exécute $U(\langle P\rangle, \langle P\rangle)$ pendant $g(|\langle P\rangle|)$ étapes
  3. répond le contraire (si $U(\langle P\rangle, \langle P\rangle)$ n'a pas terminé alors rejeter)
- Par l'absurde, si $D \in \mathrm{DTIME}(f(n))$ avec un programme $F$ alors

$F(\langle F\rangle)$ ? $U(\langle F\rangle, \langle F\rangle)$ s'arrête en $\leq g(|\langle F\rangle|)$ étapes car $f(n)\log f(n) \in o(g(n))$ 


- accepte $\Longrightarrow_{\text{(définition de } D)}$ $U(\langle F\rangle, \langle F\rangle) = F(\langle F\rangle)$ rejette ↯
- rejette $\Longrightarrow_{\text{(définition de } D)}$ $U(\langle F\rangle, \langle F\rangle) = F(\langle F\rangle)$ accepte ↯

□

**Corollaire.** $\forall k \in \mathbb{N} : \mathrm{DTIME}(n^k) \subsetneq \mathrm{DTIME}(n^{k+1})$.

**Théorème de hiérarchie en espace.** si $f(n) \in o(g(n))$ alors $\mathrm{DSPACE}(f(n)) \subsetneq \mathrm{DSPACE}(g(n))$

# La classe de complexité P

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathrm{DTIME}(n^k)$$

$$= \text{ensemble des problèmes résolus en temps polynomial } \mathcal{O}(n^k).$$

↪ avec un $k$ pour chaque algorithme

Thèse d'invariance : P est identique pour tous les modèles de calcul raisonnables.

Classe contenant de nombreux problèmes :
Entiers : addition, multiplication, exponentiation, test de primalité 😰.
Graphes : test de connexité, existence d'un couplage parfait, plus courts chemins, test de planarité, existence d'un cycle orienté pair 😰, test sans-nœud 😱.
Optimisation linéaire. Évaluation d'un circuit booléen.

P est close par : intersection, union, concaténation, complémentaire.

La composition de deux polynômes est un polynôme, donc
imbriquer des appels d'algorithmes polynomiaux reste polynomial.

**Thèse de Cobham-Edmonds.**
Les problèmes dans P sont ceux résolus par des algorithmes **efficaces**.
⚠ $3000\, n^{20}$ versus $2^{0.00001\, n}$ se croisent à environ 50 000 000… ? feasible, tractable

# La classe de complexité EXP et son intérieur...

$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}(2^{n^k})$

$=$ ensemble des problèmes résolus en temps exponentiel $\mathcal{O}(2^{n^k})$.

↪ avec un $k$ pour chaque algorithme

Classe contenant encore plus de problèmes :

- ⋆ SAT. Subset sum (entiers $e_1, \ldots, e_n$, objectif $p \in \mathbb{N}$). Set cover (ensembles $E_1, \ldots, E_n \subseteq S$, nombre $\ell \in \mathbb{N}$).
- ⋆ Bin Packing (objets $e_1, \ldots, e_n \in \mathbb{N}$, nombre de boîtes $m$ de taille $p$ en binaire).
- ⋆ Graphes : clique, stable, chemin hamiltonien ($2^n < n! < 2^{n^2}$), 3-coloration. Quantified SAT (TQBF). Super Mario Bros.
- ⋆ Évaluation d'une position du jeu des échecs, dames, go.

Théorème de hiérarchie en temps : $\mathsf{P} \subsetneq \mathsf{EXP}$.

Peut être que dans 20 ans ce cours sera totallement différent...

**Faits.** Pour de nombreux problèmes pour lesquels on ne connaît que des algorithmes en temps exponentiel dans le pire cas (ils sont donc dans EXP), **on ne sait pas** démontrer qu'ils ne sont pas dans P.

Pour les problèmes ⋆ on sait démontrer qu'ils sont dans $\mathsf{EXP} \setminus \mathsf{P}$.

$\Longrightarrow$ Néanmoins, des outils théoriques ont été développés pour en parler, dont l'importante classe des problèmes ⋆ NP-complets... trouver versus vérifier

# Les classes de complexité NP et coNP

$$\mathsf{NP} = \{L \subseteq \Sigma^* \mid \exists V \subseteq \Sigma^* \times \Gamma^* \text{ tel que } V \in \mathsf{P} \text{ et } \forall w \in \Sigma^* : w \in L \iff \exists p \in \Gamma^* : (w,p) \in V\}$$

= ensemble des problèmes où une solution peut être vérifiée en temps polynomial.

$p$ est appelé le certificat ou la preuve d'appartenance de $w$ à $L$.

$V$ est appelé le vérificateur.

Exemples :

SAT. Le certificat d'appartenance de $\varphi$ à SAT est un modèle.
Il existe un programme en temps poly pour vérifier si une valuation est un modèle. $\{(\varphi, v) \mid v \models \varphi\}$

Subset sum. Le certificat pour une instance $e_1, \ldots, e_n, p \in \mathbb{N}$ est un sous-ensemble $S \subseteq \{1, \ldots, n\}$ tel que $\sum_{i \in S} e_i = p$.
Il existe un programme en temps poly pour vérifier la somme. $\{(e_1, \ldots, e_n, p, S) \mid \sum_{i \in S} e_i = p\}$

Sudoku $n^2 \times n^2$. Vérifier une grille est facile. En revanche, trouver une solution...

**Proposition.** $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$. **Ouvert.** $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$. **Ouvert.** $\mathsf{NP} \stackrel{?}{=} \mathsf{EXP}$.

# Les classes de complexité NP et coNP

$$\text{coNP} = \{L \subseteq \Sigma^* \mid \exists V \subseteq \Sigma^* \times \Gamma^* \text{ tel que } V \in \text{P et } \forall w \in \Sigma^* : w \notin L \iff \exists p \in \Gamma^* : (w,p) \in V\}$$

$=$ ensemble des problèmes où une instance négative est facile à certifier.

$=$ complémentaire de la classe P $= \{\Sigma^* \setminus L \mid L \in \text{P}\}$

Exemples :

UNSAT. Le certificat de non-appartenance de $\varphi$ à UNSAT est un modèle.

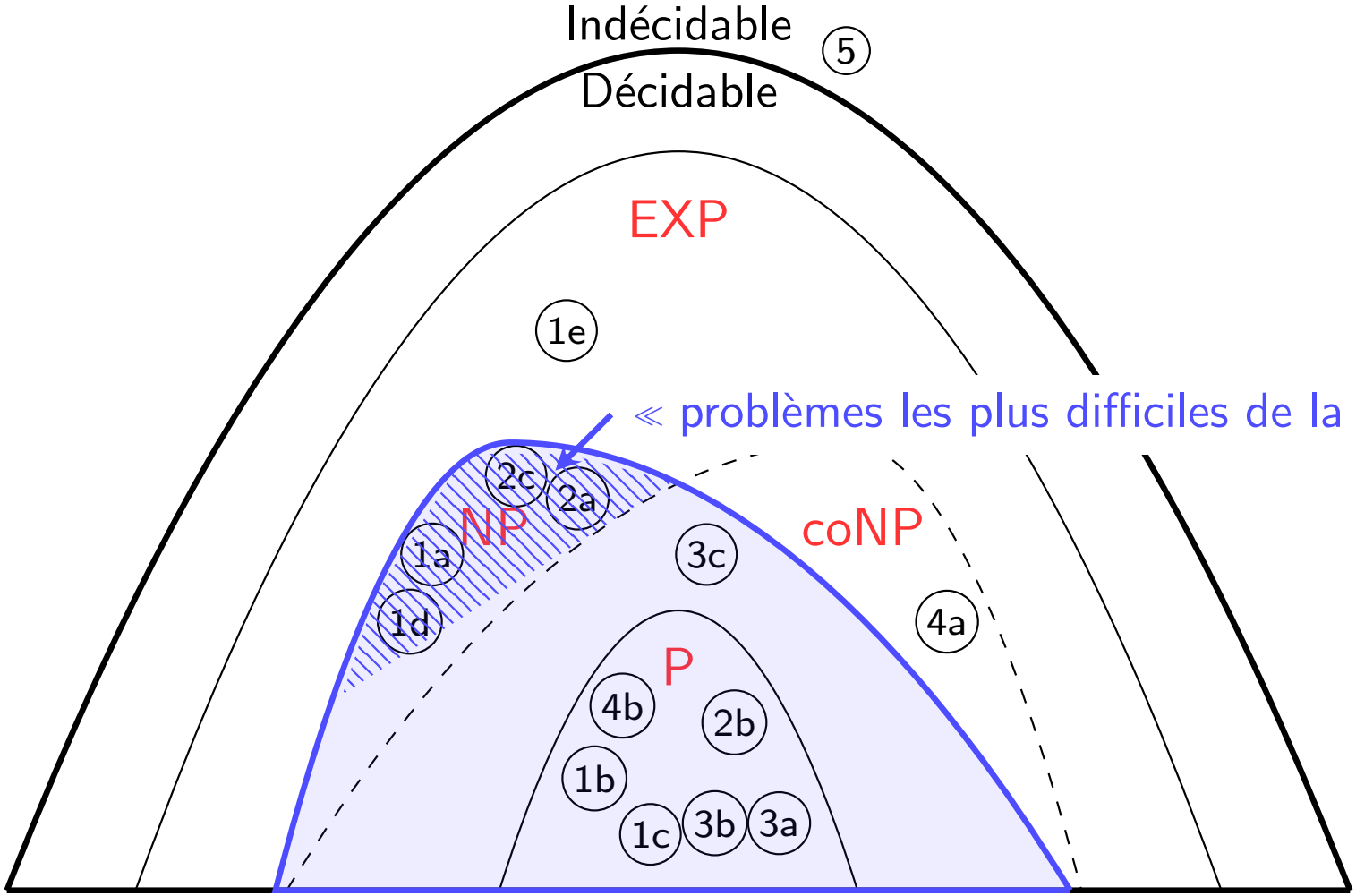
TAUTOLOGIE. Le certificat de non-appartenance de $\varphi$ à TAUTO est un contre-modèle.

$\overline{\text{PRIME}}$. Le certificat que $n \notin$ PRIME est un couple $p, q \geq 2$ tels que $n = pq$.

**Proposition.** $\text{P} \subseteq \text{NP} \cap \text{coNP}$. **Ouvert.** $\text{NP} \stackrel{?}{=} \text{coNP}$.

**Proposition.** $\text{P} = \text{NP} \implies \text{NP} = \text{coNP}$ (et la contraposée). **Preuve.** Car $\text{P} = \text{coP}$.

# Jeu : placer les problèmes suivants




1a SAT 1b 1-SAT 1c 2-SAT 1d 3-SAT 1e TQBF

2a Coloration 2b 2-Coloration 2c 3-Coloration 3a PRIME 3b $\overline{\text{PRIME}}$

3c Facteur $\leq k$ 4a $\overline{\text{Clique}}$ 4b $\overline{\text{3-Clique}}$ 5 Arrêt des MT