
Complexité : NP-complétude

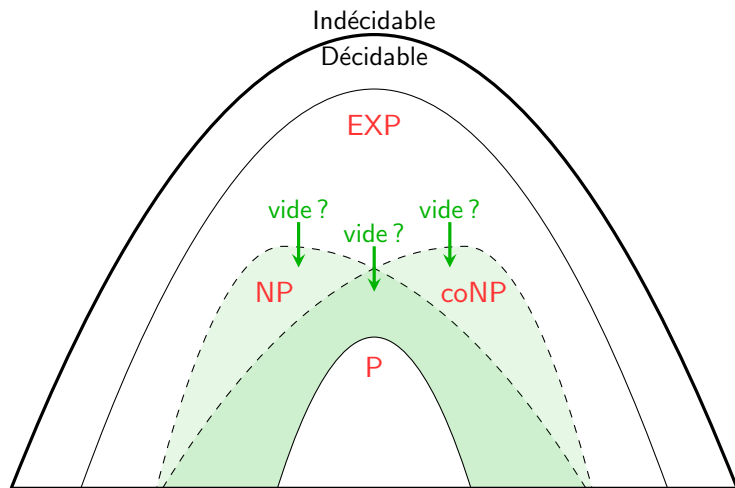
M1 Informatique Luminy 2026-27

CM-TD-TP	<u>Kévin Perrot</u>	<kevin.perrot@univ-amu.fr>
TD-TP	Antonio E. Porreca	<antonio.porreca@univ-amu.fr>
TD-TP	Ravi Kunjwal	<ravi.kunjwal@univ-amu.fr>
TP	Rocco Ascone	<rocco.ascone@lis-lab.fr>

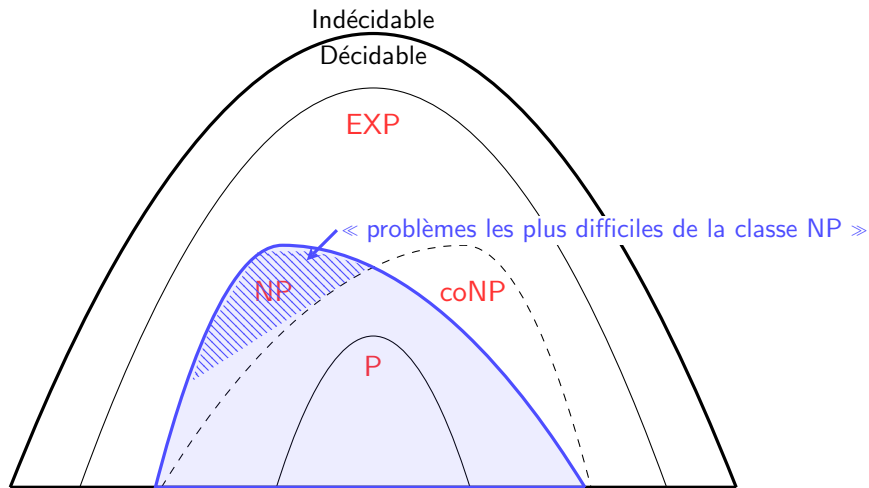
9hCM 9hTD 9hTP

$$NF = \text{MAX}(ET ; 0.3 * CC + 0.7 * ET)$$

Classes de complexité



NP-complétude



C'est la notion de **réduction** qui permet de **comparer** la complexité des problèmes.

Réduction many-one polynomiale

« Efficacement » = en temps polynomial.

Les polynômes sont clos par composition.

« B est au moins aussi difficile que A »

On peut résoudre efficacement A en utilisant un algorithme efficace pour B .

On appelle cela une **réduction polynomiale de A vers B** .

Définition. Soient $A \subseteq \Sigma_A^*$ et $B \subseteq \Sigma_B^*$. On note $A \leq_m^P B$ lorsque il existe une fonction $f : \Sigma_A^* \rightarrow \Sigma_B^*$ telle que :

$\hookrightarrow$ many-one

- f est calculable en temps polynomial.
- $\forall w \in \Sigma_A^* : w \in A \iff f(w) \in B$. La réponse à $f(w)$ pour B donne la réponse à w pour A

Lemme. Si $A \leq_m^P B$ et $B \in P$ alors $A \in P$. Preuve. $\text{Algo}_B \circ f$ résout A . $\square$

Exemple : $3\text{-SAT} \leq_m^P \text{SAT}$ car si je savais résoudre SAT efficacement, je pourrais utiliser le même algorithme pour résoudre 3-SAT efficacement (f est l'identité).

Exemple : $\text{SAT} \leq_m^P \text{Stable}$ et $\text{Stable} \leq_m^P \text{Clique}$ et $\text{Clique} \leq_m^P \text{SAT}$. 🙄

Lemme. $\leq_m^P$ est un **pré-ordre** sur les langages (problèmes).

Preuve. Réflexivité : $\forall A : A \leq_m^P A$ avec f l'identité.

Transitivité : si $A \leq_m^P B \leq_m^P C$ alors $f_{AC} = f_{BC} \circ f_{AB}$ montre que $A \leq_m^P C$:

$\forall w \in \Sigma_A^* : w \in A \iff f_{AB}(w) \in B \iff f_{BC}(f_{AB}(w)) \in C$. $\square$

$3\text{-SAT} \leq_m^P \text{Stable}$

Transformation f d'une instance de **3-SAT** en une instance de **Stable** :

Entrée : φ avec les variables $x_1, \dots, x_n$ et les clauses $C_1, \dots, C_m$.

Construire un graphe non-orienté $G = (V, E)$ avec :

1. Pour chaque clause C_j avec $1 \leq j \leq m$, ajouter un nouveau triangle à G , dont les sommets sont étiquetés par les trois littéraux de la clause.
2. Pour chacune des n variables x_i avec $1 \leq i \leq n$, ajouter une arête entre toute paire de sommets étiquetés x_i et $\neg x_i$.

Sortie : Graphe G et entier m pour la taille du stable recherché.

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

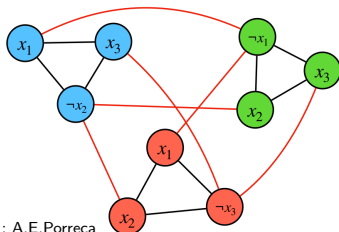


Fig : A.E.Porreca

Temps poly :

Deux boucles en $\mathcal{O}(m) + \mathcal{O}(m^2)$.

Graphe à $3m$ sommets et $\leq (3m)^2$ arrêtes.

$\Longleftrightarrow$:

Un stable de G de taille m doit contenir exactement un sommet par clause,

et ne peut pas contenir x_i et $\neg x_i$.

Donc $\exists$ stable de taille $m \implies \exists$ un modèle à φ .

Et $\exists$ un modèle à $\varphi \implies \exists$ stable de taille m .

Stable $\leq_m^P$ Clique

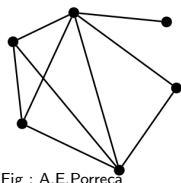
Transformation f d'une instance de **Stable** en une instance de **Clique** :

Entrée : un graphe non-orienté $G = (V, E)$ et un entier k .

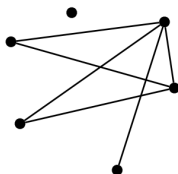
Construire le graphe complémentaire $\bar{G} = (V, V^2 \setminus E)$ sans boucles.

Sortie : Graphe $\bar{G}$ et la même valeur k pour la taille de la clique recherchée.

$G = (V, E)$



$\bar{G} = (V, V^2 - E)$



Temps poly :

Un parcours de la matrice.

$\Longleftrightarrow$:

G a un stable de taille k

$\Longleftrightarrow \bar{G}$ a une clique de taille k

$\hookrightarrow$ le même ensemble de sommets.

Fig : A.E.Porrecá

NP-complétude : définition

Définition. A est NP-difficile lorsque $\forall L \in \text{NP} : L \leq_m^P A$.
Si en plus $A \in \text{NP}$ alors A est NP-complet.

Remarque. Les problèmes NP-complets sont tous
« aussi difficiles les uns que les autres »
à un facteur polynomial près (réduction).



NP-complétude : SAT

Théorème [Cook 1971, Levin 1973]. **SAT** est NP-complet.

Preuve (idée).

1. **SAT** $\in$ **NP**. ✓

2. **SAT** est **NP-dur**, c'est-à-dire $\forall L \in \text{NP} : L \leq_m^P \text{SAT}$.

Soit $L \in \text{NP}$. On sait uniquement que :

L a un vérificateur polynomial V pour vérifier des couples instance-certificat.

Et on va en conclure que :

On peut transformer une instance de L en une instance de **SAT** avec le $\Longleftrightarrow$.

La démonstration consiste, pour une instance w de L , à construire une formule φ_w telle que :

- Une valuation x encode toute l'exécution $V(w, y)$ pour un certain y .

Technique : avec une variable par possibilité dans l'espace-temps du calcul.

- $x \models \varphi_w$ si et seulement si 1 et 2 sont vrais :

1. x est une exécution correcte de V pour vérifier le certificat y de w ,

Technique : les clauses assurent la cohérence de x avec le programme de V à chaque étape du calcul. ⚠ Partie la plus méticuleuse de la démonstration !

2. l'entrée (w, y) de cette exécution est acceptée.

Technique : une clause assure que la dernière étape du calcul accepte.

Alors : $\varphi_w \in \text{SAT} \implies \exists x : x \models \varphi_w \xrightarrow{1\&2} \exists y : V(w, y) \text{ accepte} \implies w \in L$.

$\varphi_w \notin \text{SAT} \implies \nexists x : x \models \varphi_w \xrightarrow{1\&2} \nexists y : V(w, y) \text{ accepte} \implies w \notin L$.

$\hookrightarrow$ aucun y ne certifie l'appartenance de w à L .

Comme l'espace-temps utilisé par V est de taille polynomiale, la formule obtenue est de taille polynomiale et elle est construite en temps polynomial. $\square$

NP-complétude : conséquences

Théorème [Cook 1971, Levin 1973]. **SAT** est NP-complet.

Lemme. si A est NP-difficile et $A \leq_m^P B$ alors B est NP-difficile.

Preuve. Pour tout $C \in \text{NP}$, on a $C \leq_m^P A \leq_m^P B$ donc aussi $C \leq_m^P B$ par transitivité. $\square$

Exemples :

SAT $\leq_m^P$ **3-SAT** donc **3-SAT** est NP-complet.

car **3-SAT** $\in \text{NP}$

★ **3-SAT** $\leq_m^P$ **Stable** donc **Stable** est NP-complet.

car **Stable** $\in \text{NP}$

★ **Stable** $\leq_m^P$ **Clique** donc **Clique** est NP-complet.

car **Clique** $\in \text{NP}$

Théorème. Les affirmations suivantes sont équivalentes :

1. $P = \text{NP}$.
2. Tous les problèmes NP-complets sont dans P .
3. Il existe un problème NP-complet dans P .

Remarque. La contraposées de $1 \implies 2$ est aussi pertinente.

Preuve. $1 \implies 2 \implies 3$ est immédiat, et $3 \implies 1$ car...

soit A NP-complet dans P alors tout problème de NP peut être réduit à A . $\square$

La question

Ouvert. Est-ce que $P \neq NP$?



1 000 000 \$ Clay Mathematics Institute

Possible de **vérifier efficacement** $\implies$ Possible de **trouver efficacement** ?
 $\Longleftarrow$ est évident

Est-ce que **trouver** n'est **fondamentalement** pas plus difficile que **vérifier** ?

$\hookrightarrow$ en vidéo (10') : P vs. NP and the Computational Complexity Zoo

Si $P = NP$:

Notre capacité à résoudre de nombreux problèmes passe bien à l'échelle.

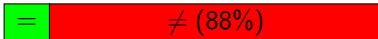
Mais comment faire de la cryptographie ? Pas de fonctions *one-way*.

Si $P \neq NP$:

De nombreux problèmes d'optimisation ne passent pas à l'échelle.

$\hookrightarrow$ comme actuellement, mais c'est démontré donc certain !

« $P = ? NP$ Poll » par Gasarch en 2019 :



coNP-complétude

A est **coNP-difficile** lorsque $\forall L \in \text{coNP} : L \leq_m^P A$.

Si en plus $A \in \text{coNP}$ alors A est **coNP-complet**.

Théorème. Si A est **NP-complet**, alors $\overline{A}$ est **coNP-complet**.

Preuve. $\overline{A} \in \text{coNP}$ par définition, et si $B \leq_m^P A$ alors $\overline{B} \leq_m^P \overline{A}$ (même transformation). $\square$

Corollaire. **UNSAT** et **TAUTO** sont coNP-complets pour $\leq_m^P$.

Preuve. UNSAT : immédiat par le théorème qui précède car $\text{UNSAT} = \overline{\text{SAT}}$.

TAUTO : $\text{UNSAT} \leq_m^P \text{TAUTO}$ en réduisant une instance φ à $f(\varphi) = \neg\varphi$. $\square$

Théorème (bis). Si $\text{NP} \neq \text{coNP}$ alors $\text{P} \neq \text{NP}$.

Preuve. $\text{P} = \text{coP}$ donc si $\text{P} = \text{NP}$ alors $\text{NP} = \text{coNP}$. $\square$

$\hookrightarrow$ prouver $\text{NP} \neq \text{coNP}$ donne



Pour aller plus loin.

Lire Karp, *Reducibility Among Combinatorial Problems* (1972).

$\hookrightarrow$ voir le diagramme page 12 du pdf.

Réduction de Cook/Turing

On peut aussi considérer des réductions autres que « many-one ».

On note $A \leq_T^P B$ lorsque il existe un **algorithme qui résoud A** en utilisant :

- un **nombre polynomial d'appels** à un algorithme qui résoud B , et
- un **temps polynomial** en dehors de ces appels.

↪ Plus souple : plusieurs appels à A sont possibles.

Proposition. Tout problème NP-dur est coNP-dur pour les réductions $\leq_T^P$.

Preuve. Soit A un problème NP-dur. Pour tout $B \in \text{coNP}$, le complémentaire $\overline{B} \in \text{NP}$, donc $\overline{B} \leq_T^P A$, et $B \leq_T^P \overline{B}$ en inversant la réponse. $\square$

⇒ Les réductions $\leq_T^P$ ne sont pas sensibles à la différence entre NP et coNP.

Ouvert. Est-ce que les réductions $\leq_m^P$ et $\leq_T^P$ définissent la même NP-complétude ?