Complexité : NP-complétude

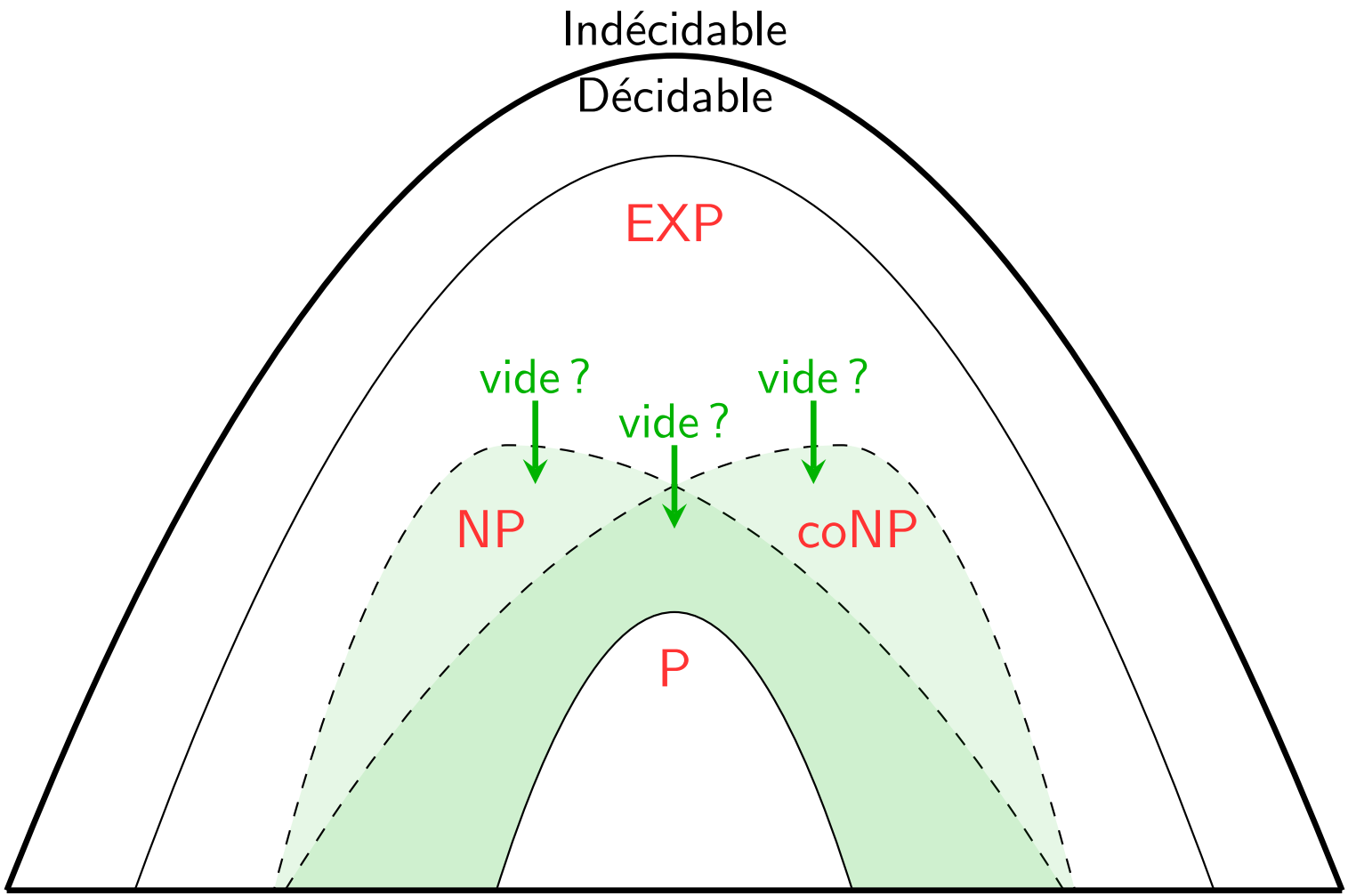
M1 Informatique Luminy 2025-26

CM-TD-TP	Kévin Perrot	<kevin.perrot@univ-amu.fr>
TD-TP	Antonio E. Porreca	<antonio.porreca@univ-amu.fr>
TD-TP	Marius Rolland	<marius.rolland@univ-amu.fr>
TP	Pablo Concha-Vega	<pablo.concha-vega@lis-lab.fr>

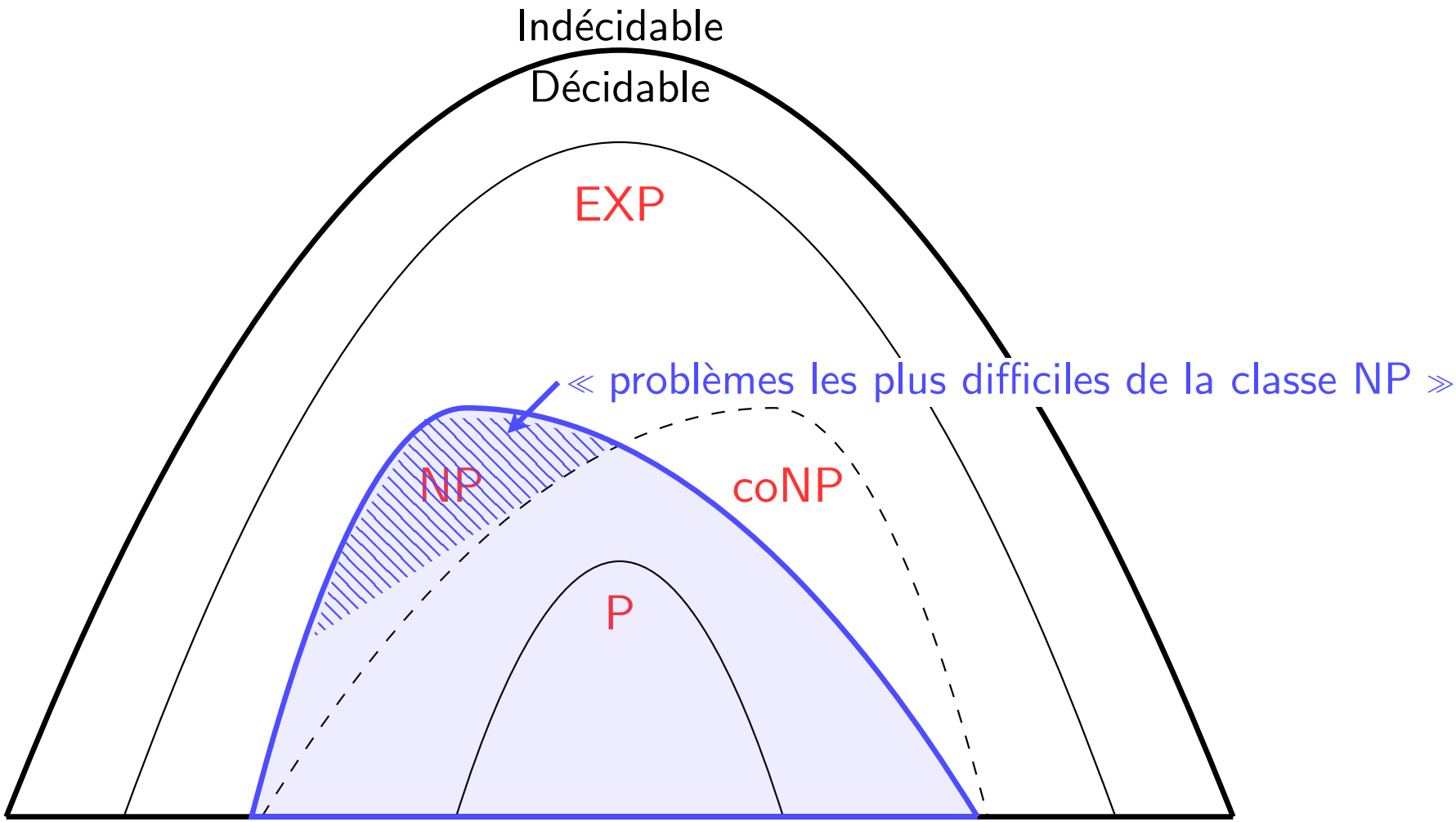
9hCM 9hTD 9hTP

NF=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.

NP-complétude : définition

Un problème A est NP-difficile ou NP-dur lorsque :

- Il est parmi les plus difficiles de la classe NP.
- Il est au moins aussi difficile que tout autre problème de NP.
- Si je sais résoudre A efficacement,
  alors je sais résoudre tous les problèmes de NP efficacement.
  $\hookrightarrow$ en temps polynomial.

Expliquer comment résoudre efficacement $B \in \text{NP}$ en utilisant un algorithme efficace pour A s'appelle une réduction polynomiale de B vers A.

On note $B \leq_T^{\text{P}} A$ lorsque « A est au moins aussi difficile que B »
il existe un algorithme qui résoud B en utilisant :

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

Exemple : 3-SAT $\leq_T^{\text{P}}$ SAT car si je savais résoudre SAT efficacement, je pourrais utiliser le même algorithme pour résoudre 3-SAT efficacement.

Exemple : 3-Clique $\leq_T^{\text{P}}$ Clique. Exemple : SAT $\leq_T^{\text{P}}$ Clique et Clique $\leq_T^{\text{P}}$ SAT. 🤔

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

?

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 $\forall B \in \text{NP} : B \leq_T^{\text{P}}$ **SAT**.

Soit $B \in$ NP. On sait uniquement que :
B a un vérificateur poly V_B pour vérifier des couples instance-certificat.
Et on va en conclure que :
On peut résoudre B en construisant une formule et demandant si elle est dans **SAT**.

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

- Une valuation x encode toute l'exécution $V_B(w, p_x)$ pour un certain p_x.
  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_B pour vérifier le certificat p_x de w,
     Technique : les clauses assurent la cohérence de x avec le programme de V_B à chaque étape du calcul. ⚠ Partie la plus méticuleuse de la démonstration !
  2. l'entrée (w, p_x) de cette exécution est acceptée.
     Technique : une clause assure que la dernière étape du calcul accepte.

Alors : $\varphi_w \in$ **SAT** $\implies \exists x : x \models \varphi_w \overset{1\&2}{\implies} \exists p_x : V_B(w, p_x)$ accepte $\implies w \in B$.

$\varphi_w \notin$ **SAT** $\implies \nexists x : x \not\models \varphi_w \overset{1\&2}{\implies} \nexists p_x : V_B(w, p_x)$ accepte $\implies w \notin B$.

↪ aucun p_x ne certifie l'appartenance de w à B.

Comme l'espace-temps utilisé par V_B est de taille polynomiale, la formule obtenue est de taille polynomiale et elle est construite en temps polynomial. □

NP-complétude : conséquences

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

Utilisation :

si B est NP-difficile et on montre $B \leq_T^P A$ alors A est NP-difficile.

En effet, un algo efficace pour A donnerait un algo efficace pour B qui donnerait un algo efficace pour tout NP.

Exemples :

SAT $\leq_T^P$ **3-SAT** donc **3-SAT** est NP-complet. car **3-SAT** $\in$ NP

⋆ **3-SAT** $\leq_T^P$ **Stable** donc **Stable** est NP-complet. car **Stable** $\in$ NP

⋆ **Stable** $\leq_T^P$ **Clique** donc **Clique** est NP-complet. car **Clique** $\in$ NP

Théorème. Les affirmations suivantes sont équivalentes :

1. P = 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...
tout problème de NP peut être réduit au problème NP-complet dans P. □

3-SAT $\leq_T^P$ Stable

On montre comment un algo poly pour **Stable** donnerait un algo poly pour **3-SAT**.

Algorithme de résolution du problème **3-SAT** :

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

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 $\neg x_i$.

Répondre comme l'algorithme pour le problème **Stable** sur l'entrée (G, m).

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

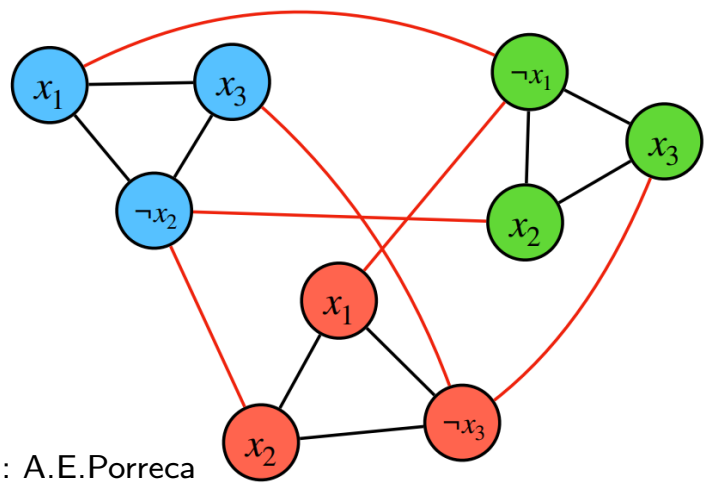


Fig : A.E.Porreca

Temps poly :

Deux boucles en $\mathcal{O}(m) + \mathcal{O}(m^2)$.
Graphe à $3m$ sommets et $\leq (3m)^2$ arrêtes,
donc temps poly et 1 appel à **Stable**.

Correction :

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.

↪ on peut montrer la contraposée

3-SAT est NP-difficile et **3-SAT** $\leq_T^P$ **Stable**, donc **Stable** est NP-difficile.

De plus **Stable** $\in$ NP, donc **Stable** est NP-complet.

Stable $\leq_T^P$ Clique

On montre comment un algo poly pour **Clique** donnerait un algo poly pour **Stable**.

Algorithme de résolution du problème **Stable** :
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.
Répondre comme l'algorithme pour le problème **Clique** sur l'entrée $(\bar{G}, k)$.

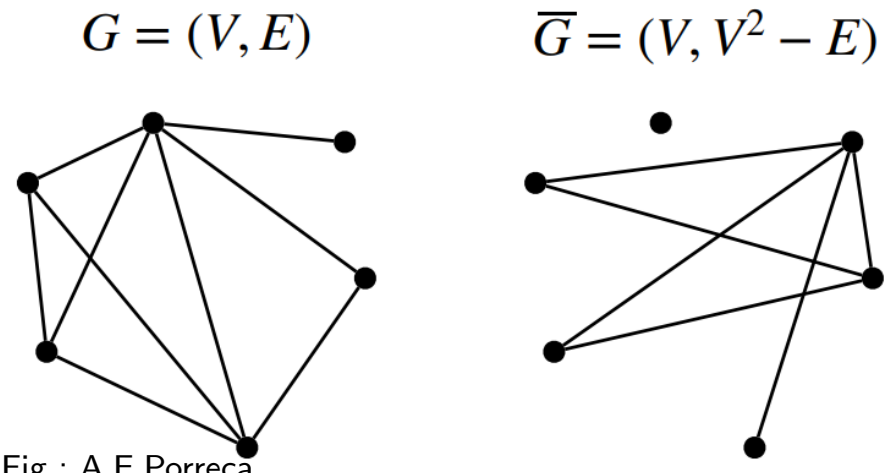


Fig : A.E.Porreca

Temps poly :
Un parcours de la matrice et 1 appel.

Correction :
G a un stable de taille k
$\iff$ $\bar{G}$ a une clique de taille k
$\hookrightarrow$ le même ensemble de sommets.

Stable est NP-difficile et **Stable** $\leq_T^P$ **Clique**, donc **Clique** est NP-difficile.
De plus **Clique** $\in$ NP, donc **Clique** est NP-complet.

La question

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

1 000 000 $ Clay Mathematics Institute

Possible de vérifier efficacement $\Longrightarrow$ Possible de trouver efficacement ?

$\Longleftarrow$ est évident

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

↪ 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.
↪ comme actuellement, mais c'est démontré donc plus besoin de chercher !

« P =? NP Poll » par Gasarch en 2019 : $=$ | $\neq$ (88%)

Un peu de théorie

Proposition. $\leq_T^{\mathrm{P}}$ est un pré-ordre sur les langages (problèmes).

Preuve. Réflexivité : $\forall A : A \leq_T^{\mathrm{P}} A$ avec une réduction triviale.

Transitivité : si $A \leq_T^{\mathrm{P}} B \leq_T^{\mathrm{P}} C$ alors on peut résoudre une instance de A en appelant un algo pour B, qui lui même appel un algo pour C.
Les polynômes sont clos par composition, donc le temps reste poly, et $A \leq_T^{\mathrm{P}} C$. □

Remarque. Les problèmes NP-complets sont tous « aussi difficiles les uns que les autres », à une composition de polynômes près (réduction).

En théorie de la complexité, on utilise davantage les réductions « many-one » polynomiales notées $A \leq_m^{\mathrm{P}} B$, qui transforment en temps polynomial une instance w de A en une instance $f(w)$ de B telle que :

$$w \in A \iff f(w) \in B$$

↪ la réponse à $f(w)$ pour B donne la réponse à w pour A.

Pour aller plus loin.

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

↪ voir le diagramme page 12 du pdf.

coNP-complétude

A est coNP-difficile lorsque $\forall B \in \text{coNP} : B \leq_T^P A$.
Si en plus $A \in \text{coNP}$ alors A est coNP-complet.

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. □

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

$\Longrightarrow$ Les réductions $\leq_m^P$ définissent une notion plus fine de coNP-complétude.

Théorème. Si A est NP-complet, alors $\overline{A}$ est coNP-complet, pour $\leq_m^P$.

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). □

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

Preuve. UNSAT : immédiat par le théorème qui précède car UNSAT$=\overline{\text{SAT}}$.
TAUTO : UNSAT $\leq_m^P$ TAUTO en réduisant une instance φ à $f(\varphi) = \neg\varphi$. □

Théorème (bis)**.** Si NP $\neq$ coNP alors P $\neq$ NP.

Preuve. P $=$ coP donc si P $=$ NP alors NP $=$ coNP. □

$\hookrightarrow$ prouver NP $\neq$ coNP donne