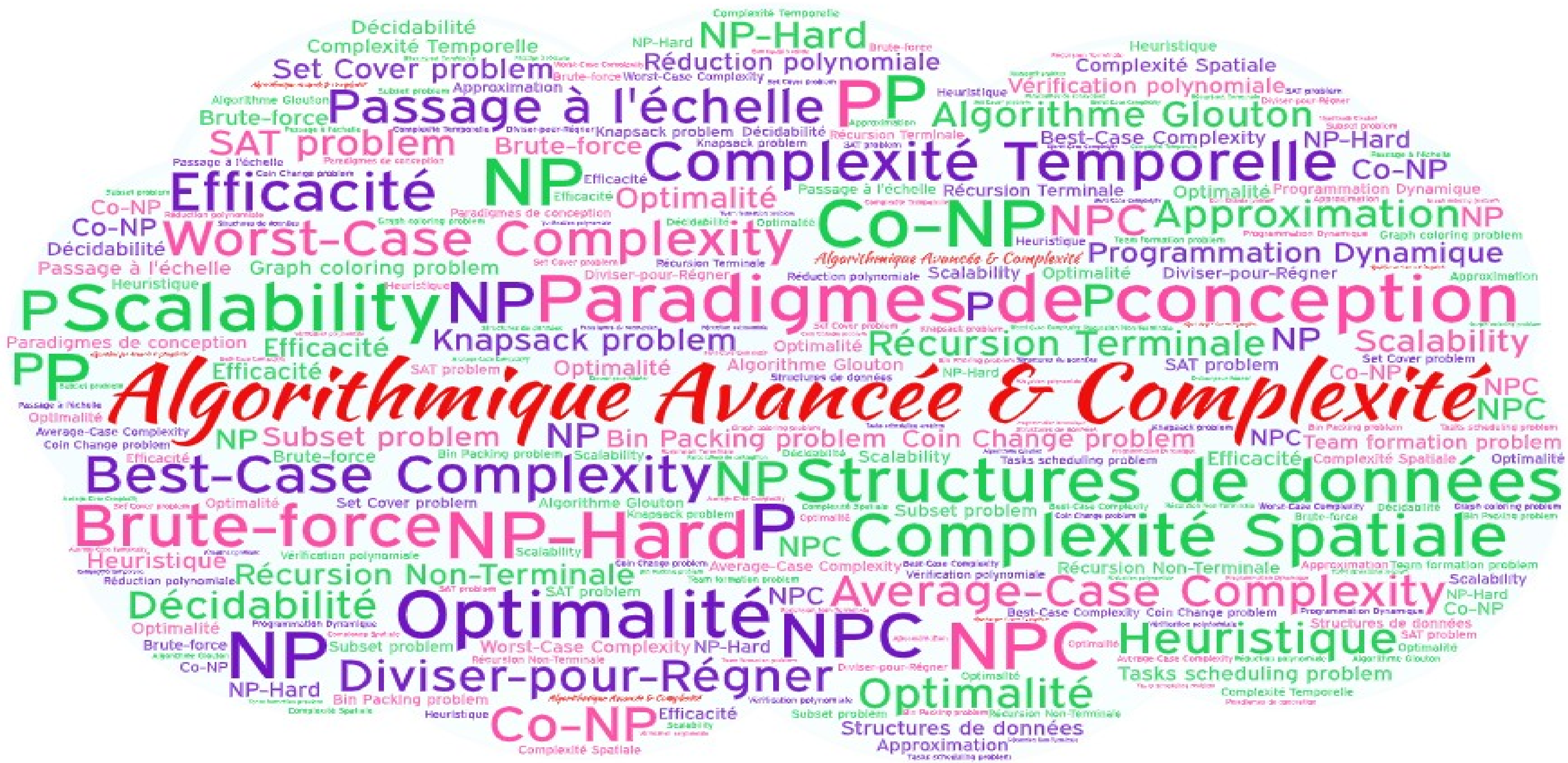


Classes de Complexité

P, NP, NPC, NP-Hard, Co-NP

Dr. Mahfoud Houari
mahfoud.houari@gmail.com
Université Abou-Bakr Belkaïd - Tlemcen

2022/2023

Classe de complexité *P* ou encore *PTime*:

- Un problème est dit dans la classe ***P*** si et seulement s'il existe un algorithme permettant de le résoudre en temps polynomial.

- La classe ***P*** contient des problèmes ayant tous des solutions *praticables* (dites aussi *raisonnables* ou *traitables*).

- Quand dit-on qu'un problème logarithmique ? Quadratique ?…

- Avantage de la classe ***P***: la composition de plusieurs solutions polynomiales donne une solution polynomiale.

- Les méthodes exactes sont propices aux problèmes de la classe ***P***.

- Certains contextes (comme le Big Data) nécessitent des approximations même pour des problèmes ***P***.

Classe de complexité P – Exemples

Déterminer si les problèmes suivants sont en P

• Accessibilité entre deux villes dans un graphe	est en P
• Vérifier si un graphe est cyclique	est en P
• Vérifier une solution Sudoku	est en P
• Exécuter une requête SQL avec K jointures	est en P
• Former le minimum d'équipes sans conflits	n'est pas en P
• Disponibilité des enseignants pour une réunion	est en P
• Horaire d'une réunion qui maximise les disponibilités	est en P
• Installation des relais téléphoniques	n'est pas en P
• Rendu de monnaie	n'est pas en P
• Détection de plagiat	n'est pas en P
• Sac à Dos	n'est pas en P

Classe de complexité *NP* :

***Définition historique* – Moins utilisée**

- ***NP*** est l'abréviation de « *N*on-déterministe *P*olynomial ».
- Classe de problèmes pouvant être décidés par une machine de Turing ***Non-déterministe*** en temps ***Polynomial***.

***Définitions alternatives* – Plus utilisée**

Un problème est ***NP*** si on peut ***proposer*** une solution candidate (pas forcément la plus optimale possible) en temps ***polynomial*** et on peut la ***vérifier*** aussi en temps ***polynomial***.

Questions capitales pour la compréhension de NP

⇒ ***Pourquoi parle t-on de proposition au lieu de résolution ?***

⇒ ***D'où vient la notion du non-déterminisme ?***

Classe de complexité *NP* – Exemple typique

Problème de Sudoku

⇒ Étant donnée une matrice de 9.9 peu remplie (*Niveau dur*).

⇒ Une solution *Sudoku* ne peut pas être trouvée en temps polynomial vu le nombre exponentiel de combinaisons à traiter.

⇒ Par contre, une fois l'utilisateur fini à jouer, la matrice remplie par lui (appelée *solution candidate*) peut être vérifiée en temps polynomial.

⇒ D'où, le *Sudoku* est bien un problème ***NP***.

Classes de complexité *P* et *NP* :

Remarques:

- ***P*** = { *Problèmes résolubles en temps polynomial* }

- ***NP*** = { *Problèmes vérifiables en temps polynomial* }

- Les classes de complexités (***P*** et ***NP***) concernent plutôt les problèmes de décision et pas les problèmes d'optimisation.

- Chaque problème d'optimisation peut être transformé en problème de décision.

Problème d'optimisation ⇒ Problème de décision

Problème d'optimisation :

Soit un graphe **G(*N*, *A*)**, et deux sommets ***u*** et ***v***. Le problème est de trouver le plus court chemin entre ***u*** et ***v*** en termes d'arêtes.

⇒ ***Résultat :*** Chemin entre ***u*** et ***v*** ayant une longueur minimale.

Problème de décision :

Soit un graphe **G(*N*, *A*)**, et deux sommets ***u*** et ***v***. Le problème est de savoir s'il existe un chemin entre ***u*** et ***v*** qui a ***k*** arêtes.

⇒ ***Résultat :*** ***Oui*** ou ***Non***.

Équivalence : Si un problème d'optimisation est ***facile*** (***difficile***) alors son problème de décision associé est aussi ***facile*** (***difficile***).

Vérification polynomiale

L'objectif est de prouver que toute solution peut être *vérifiée en temps polynomial*.

Pour cela, il faudra :

1) *Formaliser la solution (appelée aussi certificat).*
2) *Proposer un algorithme capable de vérifier la correction de toute solution du problème*
3) *Montrer que l'algorithme s'exécute en temps polynomial.*

Vérification polynomiale – Exemple 1

Définition :

Un cycle hamiltonien est un cycle simple qui passe par tous les sommets du graphe. Un ***graphe est hamiltonien*** s'il possède un cycle hamiltonien.

Problème de décision :

Vérifier si un graphe donné ***G*** est hamiltonien.

Le problème est-il en *P* ?

⇒ *Pour **N** sommets, il existe **N !** permutations possibles à tester.*

⇒ *Le problème ne possède toujours pas une solution polynomiale.*

Vérification polynomiale – Exemple 1

Définition :

Un cycle hamiltonien est un cycle simple qui passe par tous les sommets du graphe. Un ***graphe est hamiltonien*** s'il possède un cycle hamiltonien.

Problème de décision :

Vérifier si un graphe donné ***G*** est hamiltonien.

Le problème est-il en *NP* ?

*Soit un graphe **G(N, A)** et une solution candidate :* $\boldsymbol{n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_K}$.
Vérifier qu'il s'agit d'un cycle hamiltonien est peut être fait en :

$$O((K-1).|A| + K.|N|)$$

Vérification polynomiale – Exemple 1

Définition :

Un cycle hamiltonien est un cycle simple qui passe par tous les sommets du graphe. Un ***graphe est hamiltonien*** s'il possède un cycle hamiltonien.

Problème de décision :

Vérifier si un graphe donné ***G*** est hamiltonien.

Le problème est-il en *NP* ?

*Soit un graphe **G(N, A)** et une solution candidate :* $n_1 \to n_2 \to \ldots \to n_K$.
Vérifier qu'il s'agit d'un cycle hamiltonien est peut être fait en :

$$O((K-1).|A| + K.|N|) = O((K-1).|N|^2 + K.|N|) = O(|N|^3)$$

*⇒ Le problème est vérifiable en temps polynomiale ⇒ Il est en **NP**.*

Vérification polynomiale – Exemple 2

Problème de décision:

Étant donné un ensemble d'entiers $A = \{c_1, \dots, c_N\}$ et une cible C.
Existe-t-il un sous-ensemble de A dont la somme est égale à C.

Exemples :

- $A = \{2, 6, 6, 8, 9, 12\}$, $C = 25$ ⇒ Solution = {2, 6, 8, 9}.
- $A = \{2, 6, 6, 8, 9, 12\}$, $C = 13$ ⇒ Pas de solution.

Exemple d'application :

- Crypter un mot de passe.

Vérification polynomiale – Exemple 2

Problème de décision:

Étant donné un ensemble d'entiers $A = \{c_1, \ldots, c_N\}$ et une cible C.
Existe-t-il un sous-ensemble de A dont la somme est égale à C.

Le problème est-il en *P* ?

⇒ *Il existe un nombre exponentiel de cas à traiter.*
⇒ *Toujours pas de solution polynomiale*

Le problème est-il en *NP* ?

Vérification polynomiale – Exemple 2

Problème de décision:

Étant donné un ensemble d'entiers $A = \{c_1, \ldots, c_N\}$ et une cible C.
Existe-t-il un sous-ensemble de A dont la somme est égale à C.

Le problème est-il en *P* ? *NON*

Le problème est-il en *NP* ?

⇒ ***Certificat :*** *un sous-ensemble **B** de **A**.*

⇒ ***Règles à vérifier :*** $B \subseteq A$ et $\Sigma_{e \in B}\, e = C$.

⇒ ***Complexité de la vérification :*** $O(|B|.|A|+|B|)$

⇒ ***Résultat :*** *Le problème est en **NP**.*

Vérification polynomiale – Exemple 3

Problème de décision:

Factorisation en nombres premiers :
Étant donné un entier E, tester s'il existe k entiers premiers $\{n_1,\ldots,n_k\}$ tel que $E = n_1 * \ldots * n_k$.

Le problème est-il en *P* ? *NON*

Le problème est-il en *NP* ?

⇒ ***Certificat :*** *Des entiers* $\{n_1,\ldots,n_k\}$.

⇒ ***Règles à vérifier :*** *Chaque* n_i *est premier*
Leur produit donne E.

⇒ ***Complexité de la vérification :*** ***O(???)***
⇒ ***Résultat :*** ***???.***

Vérification polynomiale – Exemple 3

Problème de décision:

Factorisation en nombres premiers :
Étant donné un entier E, tester s'il existe k entiers premiers $\{n_1,\dots,n_k\}$ tel que $E = n_1 * \dots * n_k$.

Le problème est-il en *P* ? *NON*

Le problème est-il en *NP* ?

⇒ ***Certificat :*** *Des entiers* $\{n_1,\dots,n_k\}$.

⇒ ***Règles à vérifier :*** *Chaque* n_i *est premier*
Leur produit donne E.

⇒ ***Complexité de la vérification :*** $O(\Sigma_i \sqrt{n_i} + k)$

⇒ ***Résultat :*** *Le problème est en* ***NP***.

Vérification polynomiale – Exemple 4

Problème de décision:

Étant donné un ensemble d'entiers $A = \{c_1, \ldots, c_N\}$ et une cible C.

Tester s'il n'existe aucun sous-ensemble de A dont la somme est égale à C.

Le problème est-il en *P* ? *NON*

Le problème est-il en *NP* ?

- ⇒ ***Certificat :*** *Pas de certificat !*
- ⇒ ***Son inverse est en NP***
- ⇒ ***Résultat :*** *Le problème est en* ***Co-NP***.

Vérification polynomiale – Exemple 5

Problème "remplissage de boites":

Étant donné un ensemble d'objets *O* ayant chacun un poids, et un ensemble de boites ayant toutes la même capacité *C*, l'objectif est de tester si on peut ranger tous les objets dans ***K*** boites.

Le problème est-il en *P* ? *NON*

⇒ ***Chaque objet peut être rangé dans une des K boites ;***

⇒ ***K possibilités se présentent dans l'arbre des appels récursifs pour chaque objet***

⇒ ***L'arbre contiendra K^N nœuds (possibilités) différent(e)s.***

⇒ ***Pas de solution polynomiale connue***

Vérification polynomiale – Exemple 5

Problème "remplissage de boites":

Étant donné un ensemble d'objets *O* ayant chacun un poids, et un ensemble de boites ayant toutes la même capacité *C*, l'objectif est de tester si on peut ranger tous les objets dans ***K*** boites.

Le problème est-il en NP ?

Formalisation :

⇒ Définir la classe ***Objet*** et la classe ***Boite***

⇒ ***Entrées :*** entiers ***C*** et ***K***, et un tableau **Objet[]** ***O***

⇒ ***Certificat :*** une liste **ArrayList<Boite>** ***R***

Règles à vérifier ???

Vérification polynomiale – Exemple 5

Problème "remplissage de boites":

Étant donné un ensemble d'objets *O* ayant chacun un poids, et un ensemble de boites ayant toutes la même capacité *C*, l'objectif est de tester si on peut ranger tous les objets dans ***K*** boites.

Le problème est-il en NP ?

Formalisation :

⇒ Définir la classe ***Objet*** et la classe ***Boite***

⇒ ***Entrées :*** entiers ***C*** et ***K***, et un tableau **Objet[]** ***O***

⇒ ***Certificat :*** une liste **ArrayList<Boite>** ***R***

Règles à vérifier :

1. *Tous les objets du tableau* ***O*** *sont rangés dans* ***R***
2. *Chaque objet appartient à une seule boite*
3. *Les objets de chaque boite respectent la capacité* ***C***
4. *La taille de la liste* ***R*** *est égale à* ***K***

Complexité de la vérification : **O(???)**

Vérification polynomiale – Exemple 5

Problème "remplissage de boites":

Étant donné un ensemble d'objets *O* ayant chacun un poids, et un ensemble de boites ayant toutes la même capacité *C*, l'objectif est de tester si on peut ranger tous les objets dans ***K*** boites.

Le problème est-il en NP ?

Formalisation :

⇒ Définir la classe ***Objet*** et la classe ***Boite***

⇒ ***Entrées :*** entiers ***C*** et ***K***, et un tableau **Objet[]** ***O***

⇒ ***Certificat :*** une liste **ArrayList<Boite>** ***R***

Règles à vérifier :

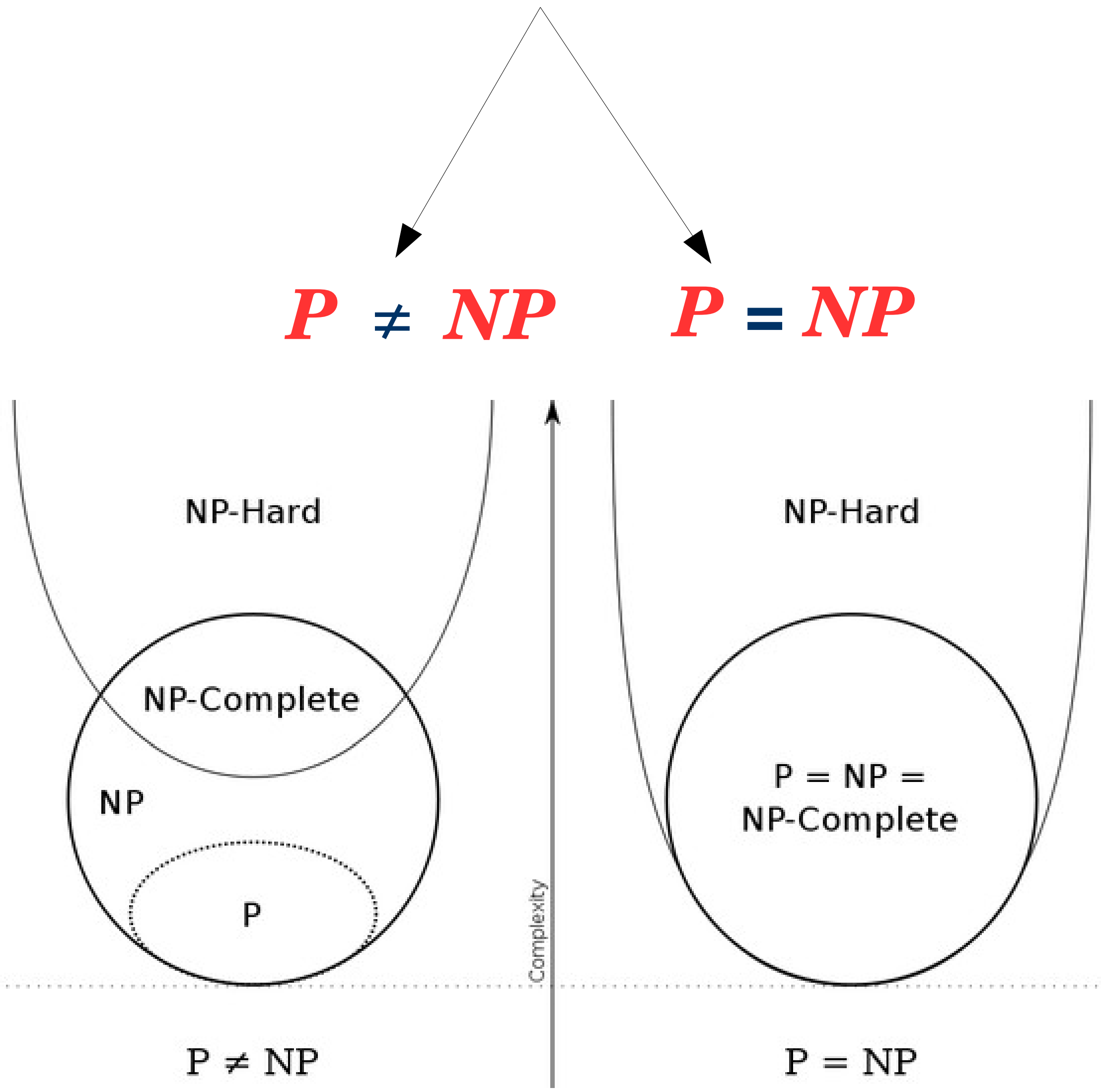
1. *Tous les objets du tableau* **O** *sont rangés dans* **R**
2. *Chaque objet appartient à une seule boite*
3. *Les objets de chaque boite respectent la capacité* **C**
4. *La taille de la liste* **R** *est égale à* **K**

Complexité de la vérification : **O(*K*.*N*)**

Résultat : le problème est vérifiable en temps polynomial

La conjecture *P ≠ NP* ?

« Pas de preuve depuis 1971 »



La conjecture $P \neq NP$?

« Pas de preuve depuis 1971 »

$P \neq NP$

Revient à montrer que l'on trouvera jamais une solution polynomiale aux problèmes NP.

$P = NP$

Revient à trouver une solution polynomiale à chaque problème NP.