Classes de Complexité

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

Dr. Mahfoud Houari

mahfoud.houari@gmail.com

Université Abou-Bakr Belkaïd - Tlemcen

2022/2023

Exemple : *Problème de Couverture Maximale*

Problème d'optimisation :

Soient donnés N ensembles $S_1,\ldots,S_N$ tel que chaque ensemble contient un ou plusieurs entiers appartenant à ensemble d'entiers U. Le but est de trouver K ensembles qui engendrent le maximum d'entiers de U.

Exemple : $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 2, 6\}$, $S_3 = \{4, 6\}$, $S_4 = \{3, 5\}$.

Pour $K = 1$: la solution optimale est $\{S_1\}$.

Pour $K = 2$: la solution optimale est $\{S_1, S_3\}$.

Pour $K = 3$: la solution optimale est $\{S_1, S_3, S_4\}$.

Exemple : *Problème de Couverture Maximale*

Problème d'optimisation :

Soient donnés N ensembles $S_1,\ldots,S_N$ tel que chaque ensemble contient un ou plusieurs entiers appartenant à ensemble d'entiers U. Le but est de trouver K ensembles qui engendrent le maximum d'entiers de U.

Exemple : $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 2, 6\}$, $S_3 = \{4, 6\}$, $S_4 = \{3, 5\}$.

Pour $K = 1$: la solution optimale est $\{S_1\}$.

Pour $K = 2$: la solution optimale est $\{S_1, S_3\}$.

Pour $K = 3$: la solution optimale est $\{S_1, S_3, S_4\}$.

Problème de décision :

Soient donnés N ensembles $S_1,\ldots,S_N$ tel que chaque ensemble contient un ou plusieurs entiers appartenant à ensemble d'entiers U. Le but est de vérifier s'il y a K ensembles qui engendrent au moins M entiers de U.

Exemple : *Problème de Couverture Maximale*

Le problème est en ***P*** ?

- C^{N}_{K} possibilités à examiner.
- Pas de solution polynomiale trouvée jusqu'à présent.

Exemple : *Problème de Couverture Maximale*

Le problème est en ***NP*** ?

- ***Certificat :***

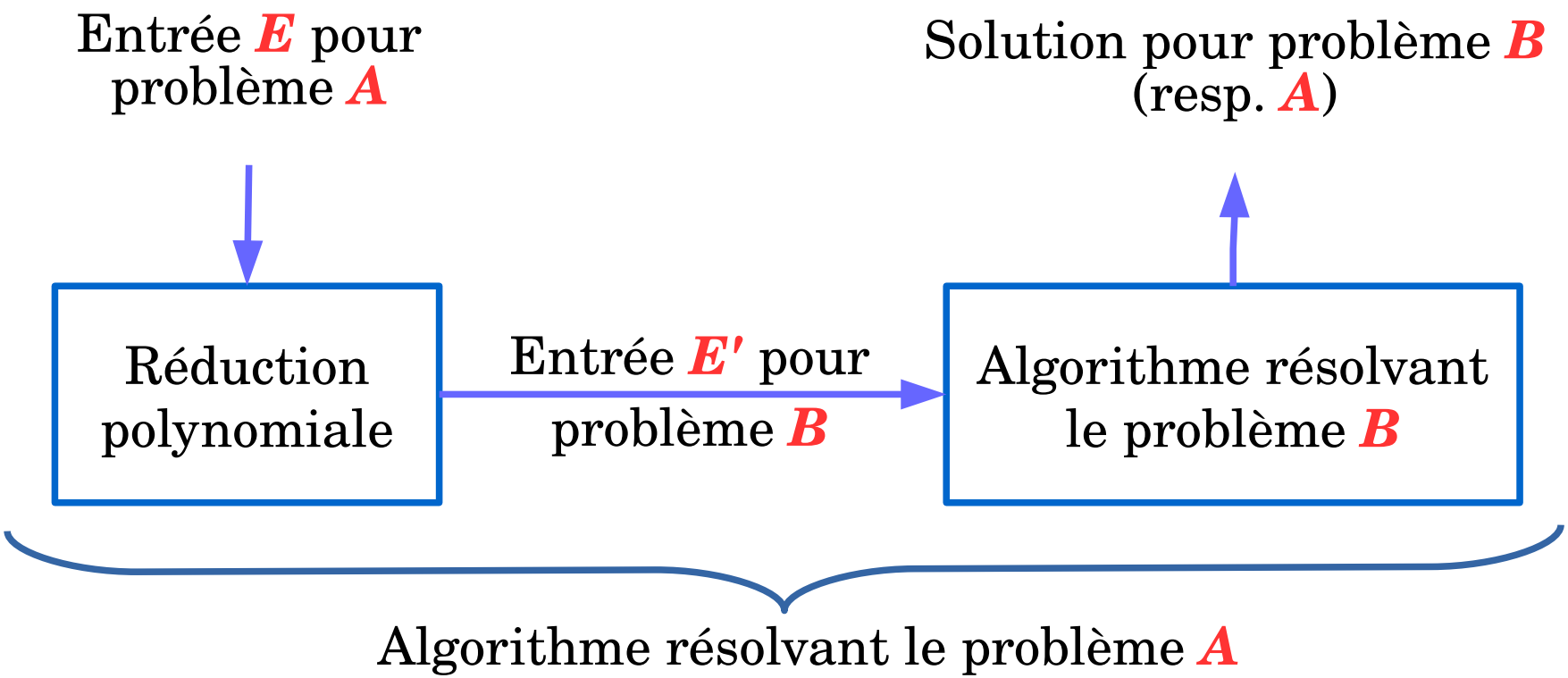
  K ensembles $S_1,\ldots,S_K$.

- ***Règles de correction :***
  - *Chaque ensemble fait partie des ensembles de départ.*
  - *Le nombre des éléments distincts* $\geq$ ***M***.
- ***Complexité de la vérification :*** $O(K.N.U + K.U)$
- ***Conclusion :***
  - Le problème est vérifiable en temps polynomial,
  - D'où, il est en ***NP***.

NP – Complétude

(*NP-Completness*)

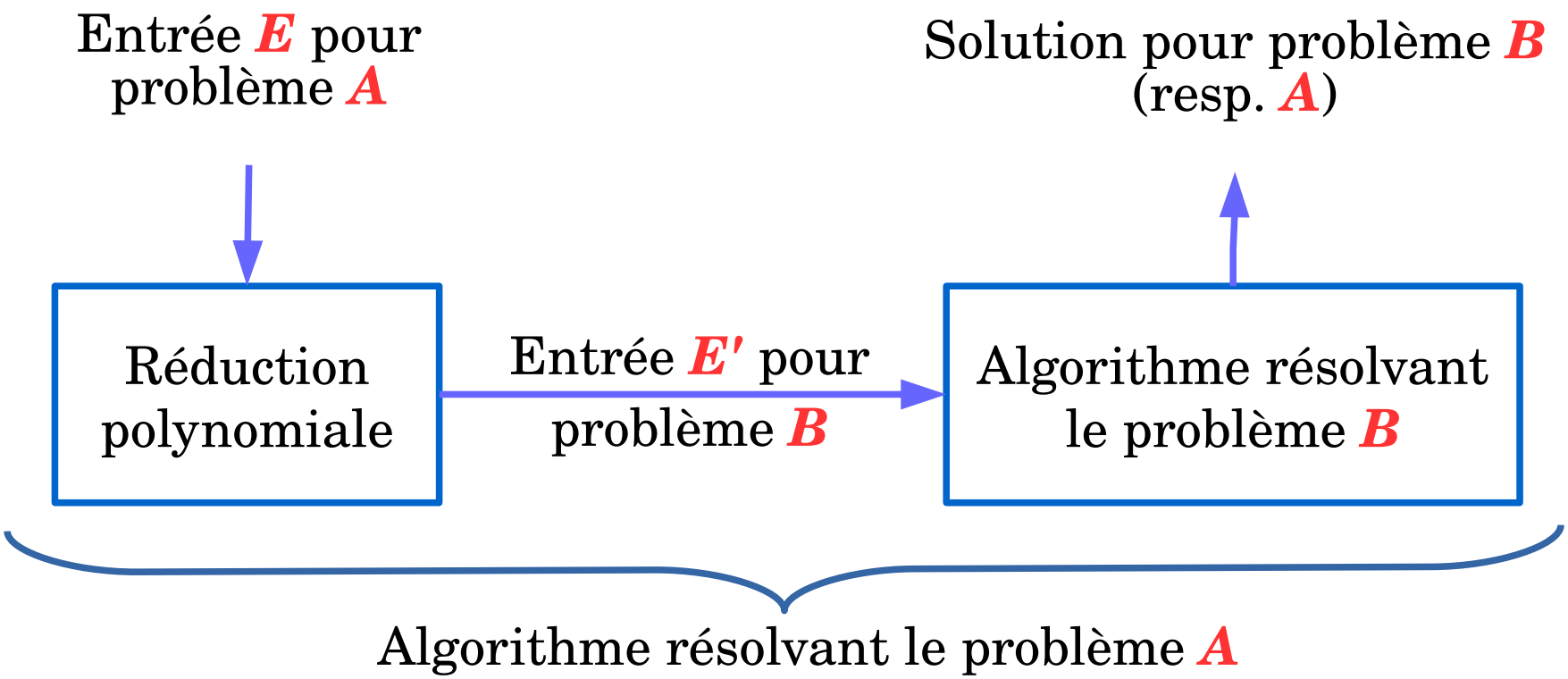
Réduction Polynomiale – Principe



Définition formelle :

Le problème ***A*** se réduit polynomialement en ***B***, noté $A \leq_p B$, s'il existe une fonction ***f*** qui transforme chaque entrée ***a*** de ***A*** en une entrée ***f(a)*** pour ***B*** tel que : ***a*** retourne *Vrai* sur le problème ***A*** si et seulement si ***f(a)*** retourne *Vrai* sur le problème ***B***.

Réduction Polynomiale – Principe



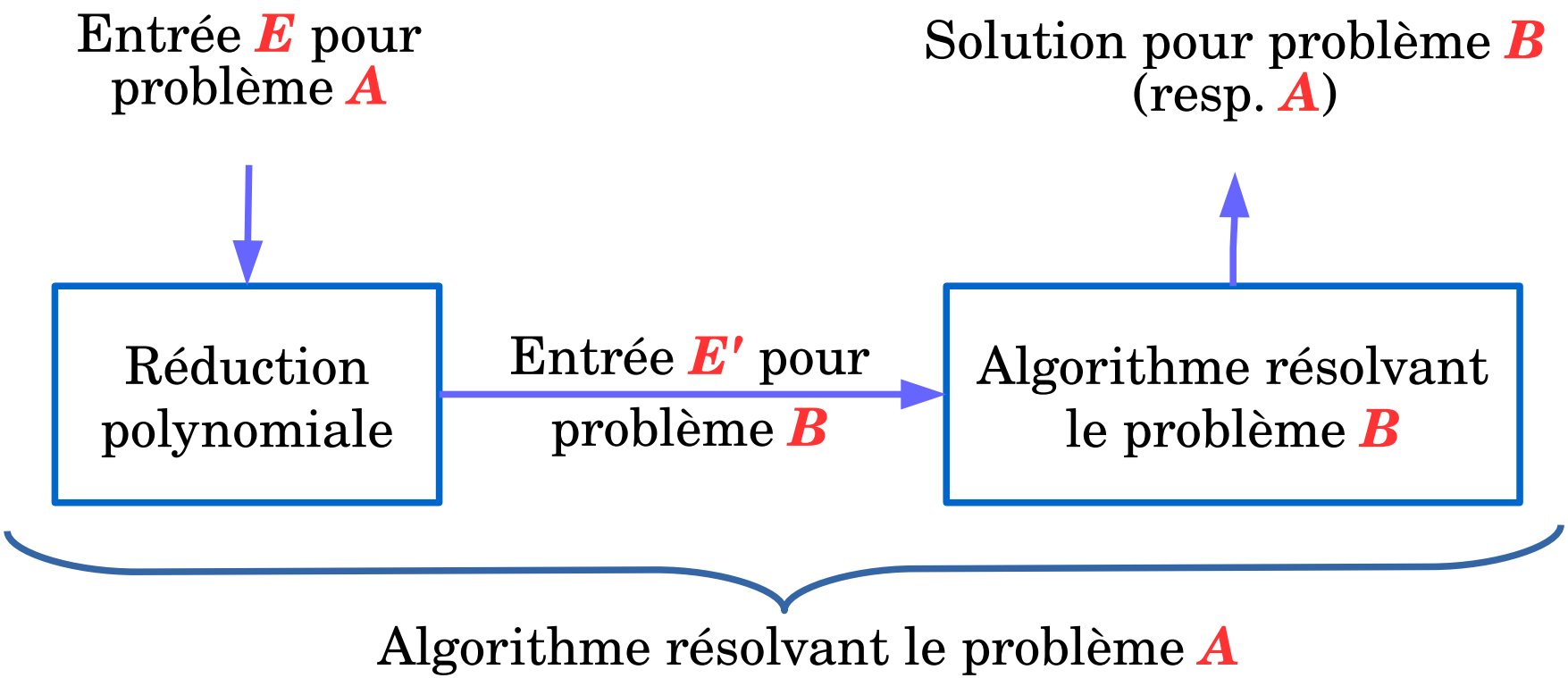
Exemple simple :

Problème B : calculer X^N. $Sol_B(X, N) = X^N$.

Problème A : calculer X^2.

Réduction de A en B : $Sol_A(X) = Sol_B(X, 2)$.

Réduction Polynomiale – Principe



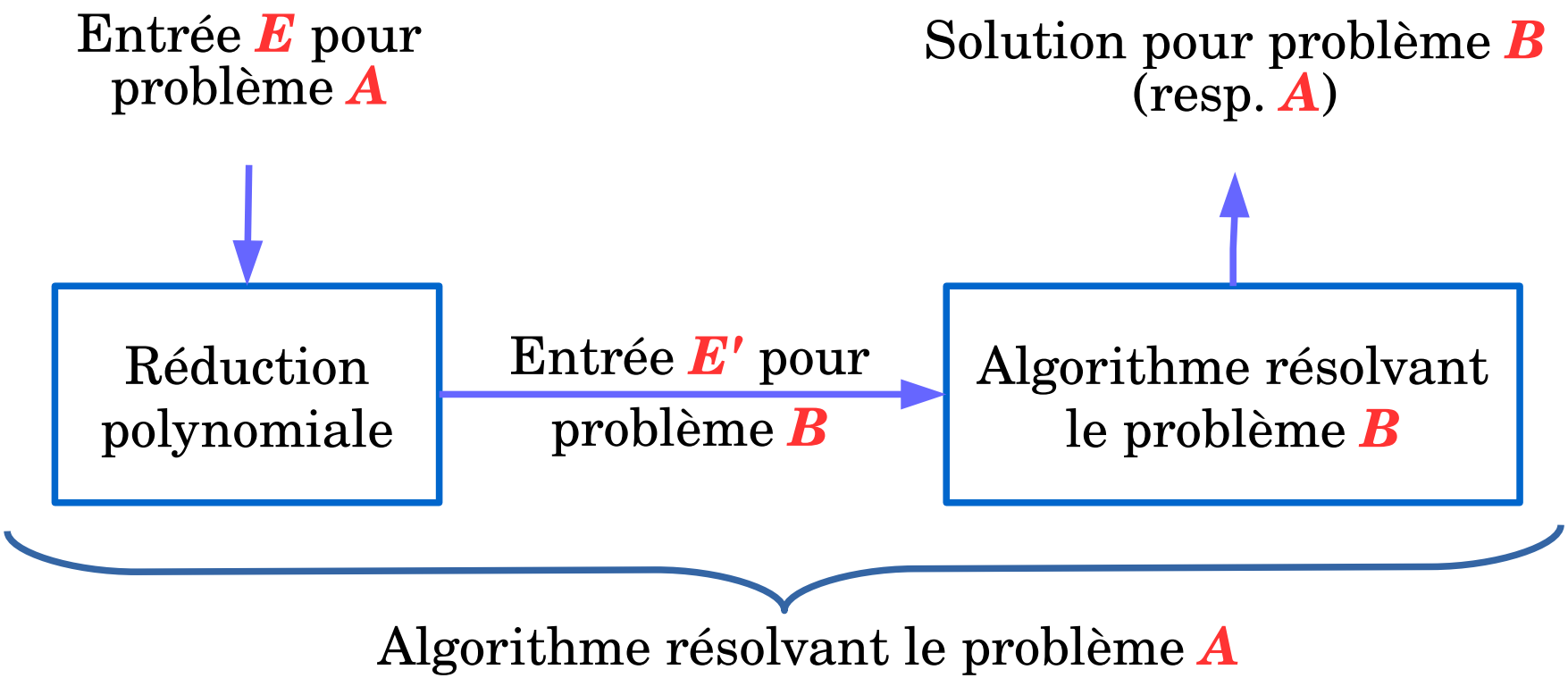
Pourquoi faire une réduction polynomiale ?

⇒ Si $A \leq_p B$ alors que le problème *A* est moins dur ou plus dur que le problème *B* ?

⇒ Si $A \leq_p B$ et *B* est en *P*, alors *A* ?

⇒ Si $A \leq_p B$ et *A* appartient à *NP*, alors *B* ?

Réduction Polynomiale – Principe



Pourquoi faire une réduction polynomiale ?

⇒ Si $A \leq_p B$ alors le problème *A* est moins dur que le problème *B*.

⇒ Si $A \leq_p B$ et *B* est en *P*, alors *A* appartiendrait aussi à *P*.

⇒ Si $A \leq_p B$ et *A* est en *NP*, alors *B* est aussi en *NP*.

Réduction Polynomiale – Principe

Théorème de transition :

Si le problème ***P1*** se réduit polynomialement en ***P2***, et ***P2*** se réduit polynomialement en ***P3***, alors on peut conclure facilement que : ***P1*** se réduit polynomialement en ***P3***.

Notation : $P1 \leq_p P2$ & $P2 \leq_p P3 \Rightarrow P1 \leq_p P3$

Importance : Ce résultat prend toute son importance lorsqu'on désire classer un nouveau problème dans la classe ***NPC***

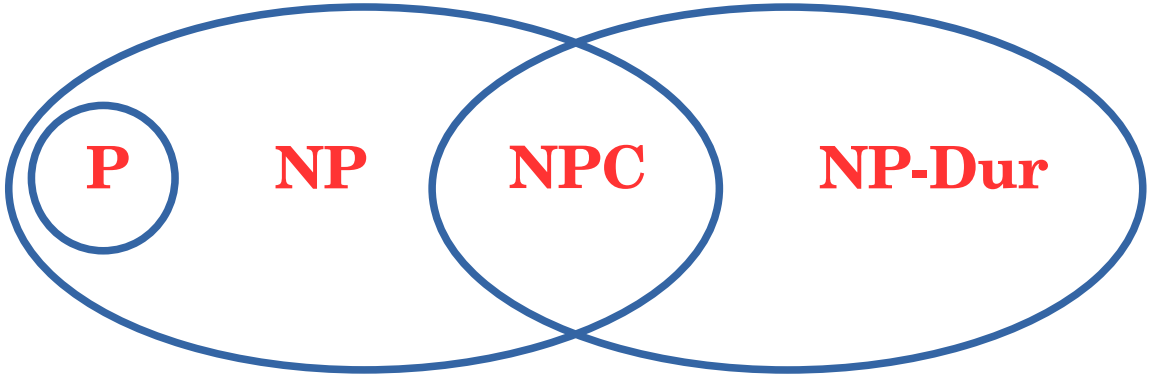
Qu'est-ce qu'un problème NP–Complet ?

Définition :

Un problème de décision ***A*** est **NP-Dur** si pour tout problème ***B*** en **NP**: $B \leq_p A$.

Un problème de décision ***A*** est **NP-Complet** si : ***A*** est **NP** et aussi bien **NP-Dur**.

On dénote par **NPC** la classe des problèmes **NP-Complets**.



Qu'est-ce qu'un problème NP–Complet ?

Définition :

Un problème de décision ***A*** est **NP-Dur** si pour tout problème ***B*** en **NP**: $B \leq_p A$.

Un problème de décision ***A*** est **NP-Complet** si : ***A*** est **NP** et aussi bien **NP-Dur**.

On dénote par **NPC** la classe des problèmes **NP-Complets**.

Comment montrer qu'un problème est **NPC** ?

Qu'est-ce qu'un problème NP–Complet ?

Définition :

Un problème de décision ***A*** est **NP-Dur** si pour tout problème ***B*** en **NP**: $B \leq_p A$.

Un problème de décision ***A*** est **NP-Complet** si : ***A*** est **NP** et aussi bien **NP-Dur**.

On dénote par **NPC** la classe des problèmes **NP-Complets**.

Comment montrer qu'un problème est **NPC** ?

Selon la définition :

Ça risque de prendre énormément de temps vu qu'il faudra vérifier la condition **(2)** de la définition pour tout problème ***NP***.

Qu'est-ce qu'un problème NP–Complet ?

Définition :

Un problème de décision ***A*** est **NP-Dur** si pour tout problème ***B*** en **NP**: $B \leq_p A$.

Un problème de décision ***A*** est **NP-Complet** si : ***A*** est **NP** et aussi bien **NP-Dur**.

On dénote par **NPC** la classe des problèmes **NP-Complets**.

Comment montrer qu'un problème est **NPC** ?

Intérêt de la réduction :

Pour tous problèmes ***A*** et ***B*** de la classe ***NP***. Si ***A*** est ***NP-Dur***, et $A \leq_p B$, alors on peut déduire que : ***B*** est aussi ***NP-Dur***.

Premier problème NP–Complet – SAT

Définition :

SAT c'est le problème de satisfiabilité de formules booléennes. Étant donné une formule de logique propositionnelle, déterminer s'il existe une valuation qui rend cette formule ***vraie***.

Exemples :

- $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$
  $\Rightarrow$ **Satisfiable: $v(x_1) = 1$, $v(x_3) = 0$.**
- $(x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_3)$
  $\Rightarrow$ **Non satisfiable :** MAIS Comment le prouver ?

Premier problème NP–Complet – SAT

Définition :

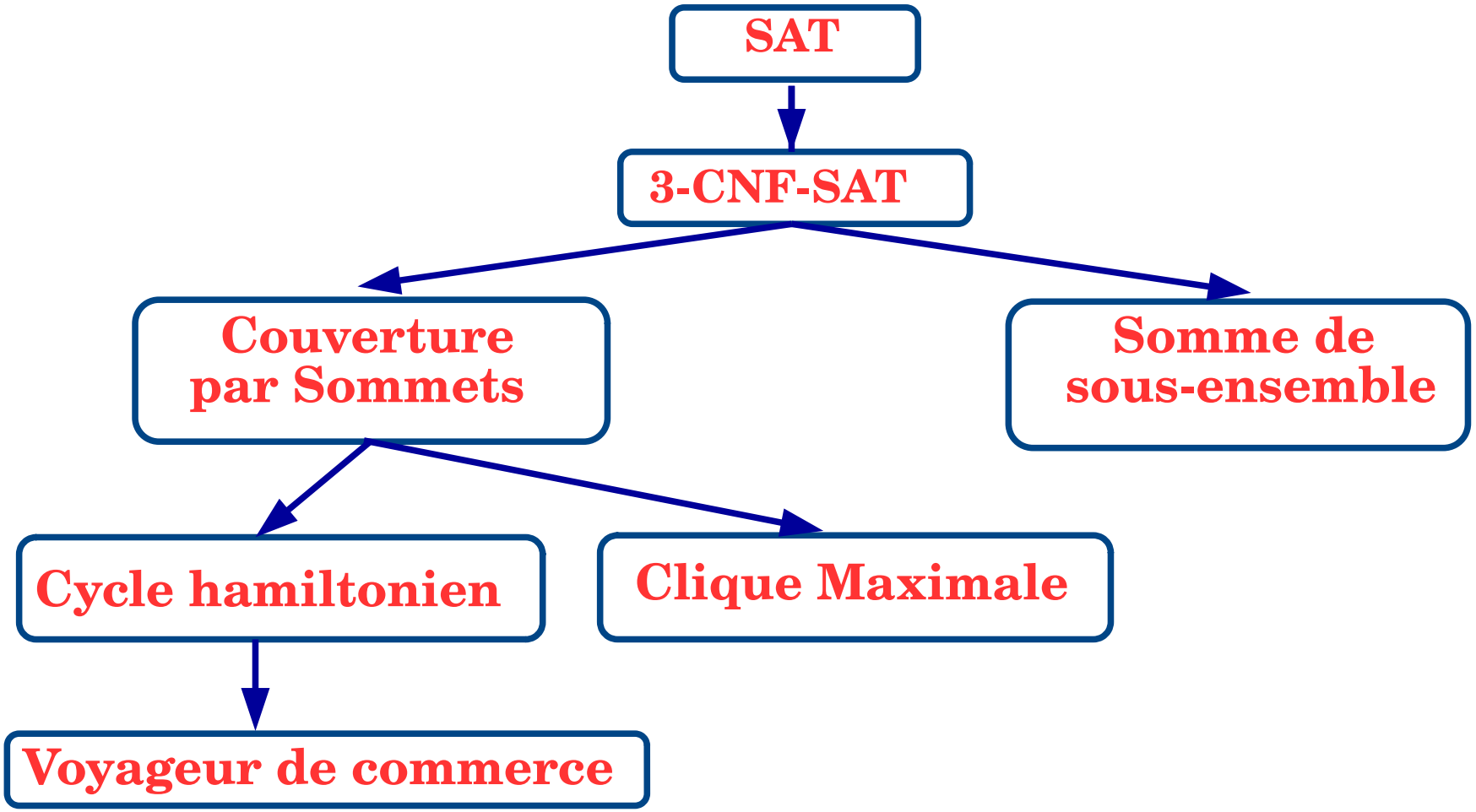
SAT c'est le problème de satisfiabilité de formules booléennes. Étant donné une formule de logique propositionnelle, déterminer s'il existe une valuation qui rend cette formule ***vraie***.

Théorème de Cook:

Le problème **SAT** *est* **NPC**.

Premier problème NP–Complet – SAT

Tous les autres problèmes **NPC** ont été démontrés directement ou indirectement par réduction polynomiale à partir de **SAT**.



Exemple de démonstration de NP-Complétude

On a une liste ***L*** de ***N*** personnes et une matrice ***C*** tel que ***C*[*i*][*j*] = 1** si la personne ***i*** et ***j*** peuvent travailler ensemble et **0** sinon.

Problème de ***K-équipes***: Peut-on former ***K*** équipes contenant chacune que de personnes qui s'entendent bien ?

But : Vérifier que ce problème est ***NPC***.

Exemple de démonstration de NP-Complétude

On a une liste ***L*** de ***N*** personnes et une matrice ***C*** tel que ***C[i][j] = 1*** si la personne ***i*** et ***j*** peuvent travailler ensemble et **0** sinon.

Problème de ***K-équipes***: Peut-on former ***K*** équipes contenant chacune que de personnes qui s'entendent bien ?

But : Vérifier que ce problème est ***NPC***.

Étapes à suivre :

1. Montrer que le problème de ***K-équipes*** est en ***NP***.
2. Montrer que ***K-coloriage*** $\leq_p$ ***K-équipes***.
3. Sachant que le problème de ***K-coloriage*** est ***NP-Hard*** (déjà prouvé), en déduire la classe du problème ***K-équipes***.

Exemple de démonstration de NP-Complétude

1. Montrer que le problème de ***K-équipes*** est en ***NP***.

Certificat : ???

Exemple de démonstration de NP-Complétude

1. Montrer que le problème de ***K-équipes*** est en ***NP***.

Certificat : une solution contenant ***K*** équipes

Formalisation :

1. *ArrayList<Equipe>* ***E***
2. *int* [] ***L*** (liste des personnes)
3. *int*[][] ***C*** (matrice de conflits)

Règles à vérifier :

????

Exemple de démonstration de NP-Complétude

1. Montrer que le problème de ***K-équipes*** est en ***NP***.

Certificat : une solution contenant ***K*** équipes

Formalisation :

1. *ArrayList<Equipe>* ***E***
2. *int* [] ***L*** (liste des personnes)
3. *int*[][] ***C*** (matrice de conflits)

Règles à vérifier :

1. Les personnes au sein de chaque équipe s'entendent bien mutuellement
2. Chaque personne est affectée à une seule équipe
3. Toutes les personnes sont affectées

Complexité de la vérification : O(???)

Exemple de démonstration de NP-Complétude

1. Montrer que le problème de ***K-équipes*** est en ***NP***.

Certificat : une solution contenant ***K*** équipes

Formalisation :

1. *ArrayList<Equipe>* ***E***
2. *int* [] ***L*** (liste des personnes)
3. *int*[][] ***C*** (matrice de conflits)

Règles à vérifier :

1. Les personnes au sein de chaque équipe s'entendent bien mutuellement
2. Chaque personne est affectée à une seule équipe
3. Toutes les personnes sont affectées

Complexité de la vérification : $O(K.N^2)$

Résultat : *le problème est vérifiable en temps polynomial.*

Exemple de démonstration de NP-Complétude

2. Montrer que ***K-coloriage*** $\leq_p$ ***K-équipes***.

Entrées du K-coloriage : un graphe **G=(N,A)**, un entier ***K*** représentant le nombre de couleurs à utiliser.

Entrées du K-équipes : une liste de personnes ***L***, une matrice de conflits ***C***, et un entier ***K*** représentant le nombre d'équipes à former.

Réduction :

1. *Les nœuds du graphe conduisent à la liste* ***L***
2. *Les arcs du graphe conduisent à la matrice* ***C***
3. *Le nombre de couleurs devient le nombre d'équipes*

Complexité de la réduction : $O(N^2)$

Exemple de démonstration de NP-Complétude

3. En déduire la classe de ***K-équipes***.

- ***K-équipes*** est ***NP***
- ***K-coloriage*** $\leq_p$ ***K-équipes***
- ***K-coloriage*** est *NP-Hard*
- *D'où, le problème de* ***K-équipes*** *est* ***NPC***

Résoudre un problème NPC

- Se contenter à une solution non-polynomiale si la taille des données est relativement petite.
- Trouver des cas particuliers du problème pouvant être résolus en temps polynomial.
- Chercher une solution ***quasi-optimale*** (approximative).