

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Un **problème** est de **complexité polynomiale** s'il existe un algorithme de complexité polynomiale le résolvant.

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Un **problème** est de **complexité polynomiale** s'il existe un algorithme de complexité polynomiale le résolvant.

Postulat de Church: Tous les modèles de calcul raisonnables sont équivalents du point de vue de la complexité polynomiale.

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Un **problème** est de **complexité polynomiale** s'il existe un algorithme de complexité polynomiale le résolvant.

Postulat de Church: Tous les modèles de calcul raisonnables sont équivalents du point de vue de la complexité polynomiale.

(machine de Turing, pseudo-codes impératifs ou fonctionnels...
pas les algo sur les réels ou les algo quantiques)

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Un **problème** est de **complexité polynomiale** s'il existe un algorithme de complexité polynomiale le résolvant.

Postulat de Church: Tous les modèles de calcul raisonnables sont équivalents du point de vue de la complexité polynomiale.

(machine de Turing, pseudo-codes impératifs ou fonctionnels...
pas les algo sur les réels ou les algo quantiques)

Indépendance via-à-vis de la représentation: on ne veut pas devoir préciser le choix exact des structures de données utilisées .

Complexité polynomiale

On s'intéresse à des problèmes décidables, dont on veut mesurer la complexité, au sens du temps de calcul dans le pire cas.

Un **algorithme** est de **complexité polynomiale** s'il existe un polynôme P tel que le nombre d'instructions élémentaires effectuées lors de son exécution sur n'importe quelle donnée de taille n soit au plus $P(n)$.

Un **problème** est de **complexité polynomiale** s'il existe un algorithme de complexité polynomiale le résolvant.

Postulat de Church: Tous les modèles de calcul raisonnables sont équivalents du point de vue de la complexité polynomiale.

(machine de Turing, pseudo-codes impératifs ou fonctionnels...
pas les algo sur les réels ou les algo quantiques)

Idée forte. En pratique, le plus souvent, polynomial \Rightarrow efficace
(parfois après quelques années d'effort)

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.
La donnée possède-t-elle la propriété suivante...

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.

La donnée possède-t-elle la propriété suivante...

On note $\text{Oui}(D)$ l'ensemble des **instances** auxquelles on répond **oui**.

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.

La donnée possède-t-elle la propriété suivante...

On note $\text{Oui}(D)$ l'ensemble des **instances** auxquelles on répond **oui**.

Recherche / décision:

- Trouver un arbre couvrant: instance = G ; réponse = un arbre couvrant
- Existence d'un arbre couvrant: instance = G ; $\text{Oui}(D) = \{G \text{ connexes}\}$.

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.

La donnée possède-t-elle la propriété suivante...

On note $\text{Oui}(D)$ l'ensemble des **instances** auxquelles on répond **oui**.

Recherche / décision:

- Trouver un arbre couvrant: instance = G ; réponse = un arbre couvrant
- Existence d'un arbre couvrant: instance = G ; $\text{Oui}(D) = \{G \text{ connexes}\}$.

Réduction de la recherche à la décision par test d'hypothèse:

- $A := \emptyset$; tant que A n'est pas couvrant répéter: soit a une arête de G ,
 - si $G \setminus \{a\} \in \text{Oui}(D)$ alors $G := G \setminus \{a\}$ (a inutile, la supprimer)
 - sinon alors $A := A \cup \{a\}$, $G := G/a$ (a nécessaire, la contracter).

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.

La donnée possède-t-elle la propriété suivante...

On note $\text{Oui}(D)$ l'ensemble des **instances** auxquelles on répond **oui**.

Recherche / décision:

- Trouver un arbre couvrant: instance = G ; réponse = un arbre couvrant
- Existence d'un arbre couvrant: instance = G ; $\text{Oui}(D) = \{G \text{ connexes}\}$.

Réduction de la recherche à la décision par test d'hypothèse:

- $A := \emptyset$; tant que A n'est pas couvrant répéter: soit a une arête de G ,
 - si $G \setminus \{a\} \in \text{Oui}(D)$ alors $G := G \setminus \{a\}$ (a inutile, la supprimer)
 - sinon alors $A := A \cup \{a\}$, $G := G/a$ (a nécessaire, la contracter).

Optimisation / décision:

- Trouver un arbre couvrant de poids minimum
- Existence d'un arbre couvrant de poids $\leq k$

Problèmes de décision vs problème d'optimisation

Un problème D est **de décision** si la réponse est binaire: **oui** ou **non**.

La donnée possède-t-elle la propriété suivante...

On note $\text{Oui}(D)$ l'ensemble des **instances** auxquelles on répond **oui**.

Recherche / décision:

- Trouver un arbre couvrant: instance = G ; réponse = un arbre couvrant
- Existence d'un arbre couvrant: instance = G ; $\text{Oui}(D) = \{G \text{ connexes}\}$.

Réduction de la recherche à la décision par test d'hypothèse:

- $A := \emptyset$; tant que A n'est pas couvrant répéter: soit a une arête de G ,
 - si $G \setminus \{a\} \in \text{Oui}(D)$ alors $G := G \setminus \{a\}$ (a inutile, la supprimer)
 - sinon alors $A := A \cup \{a\}$, $G := G/a$ (a nécessaire, la contracter).

Optimisation / décision:

- Trouver un arbre couvrant de poids minimum
- Existence d'un arbre couvrant de poids $\leq k$

Lorsque le critère d'optimisation est borné *a priori*, réduction de l'optimisation à la décision par dichotomie et test d'hypothèse

Réduction entre problèmes: premier exemple

Donnée: Un algorithme qui trouve une solution d'un système d'inéquations linéaires en temps polynomial.

Problème: Résoudre un problème de programmation linéaire.

Réduction entre problèmes: premier exemple

Donnée: Un algorithme qui trouve une solution d'un système d'inéquations linéaires en temps polynomial.

Problème: Résoudre un problème de programmation linéaire.

Utiliser la dualité:

Réduction entre problèmes: premier exemple

Donnée: Un algorithme qui trouve une solution d'un système d'inéquations linéaires en temps polynomial.

Problème: Résoudre un problème de programmation linéaire.

Utiliser la dualité:

$$\left\{ \begin{array}{l} Ax \leq b \\ \max(c \cdot x) \end{array} \right. \quad \left\{ \begin{array}{l} Ax \leq b \\ yA \geq c, y \geq 0 \\ c \cdot x \geq y \cdot b \end{array} \right.$$

Réduction entre problèmes: premier exemple

Donnée: Un algorithme qui trouve une solution d'un système d'inéquations linéaires en temps polynomial.

Problème: Résoudre un problème de programmation linéaire.

Utiliser la dualité:

$$\left\{ \begin{array}{l} Ax \leq b \\ \max(c \cdot x) \end{array} \right. \quad \left\{ \begin{array}{l} Ax \leq b \\ yA \geq c, y \geq 0 \\ c \cdot x \geq y \cdot b \end{array} \right.$$

Réduction: Si on sait résoudre les systèmes d'inéquations linéaires en temps polynomial, alors on sait aussi résoudre les problèmes de programmation linéaire.

On a **réduit polynomialement** la programmation linéaire à la résolution d'équations linéaires.

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

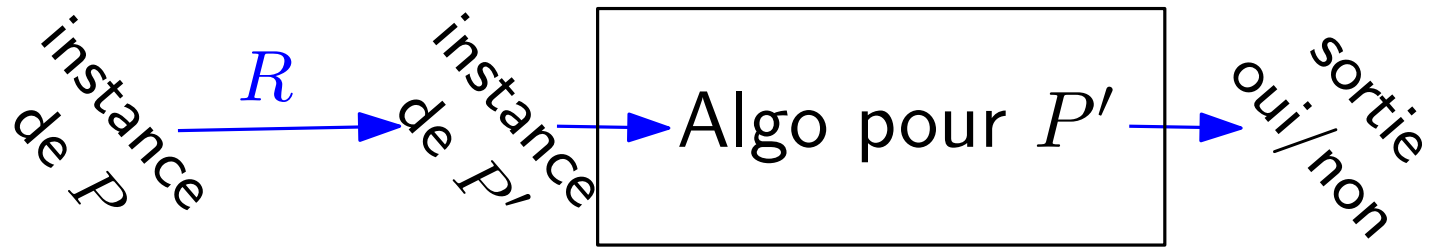
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



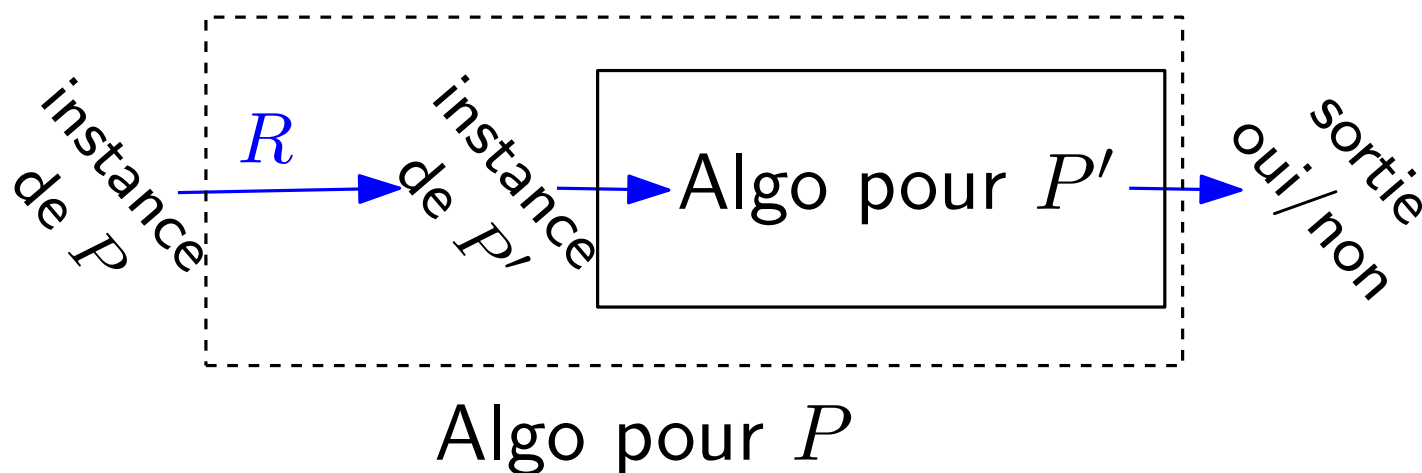
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



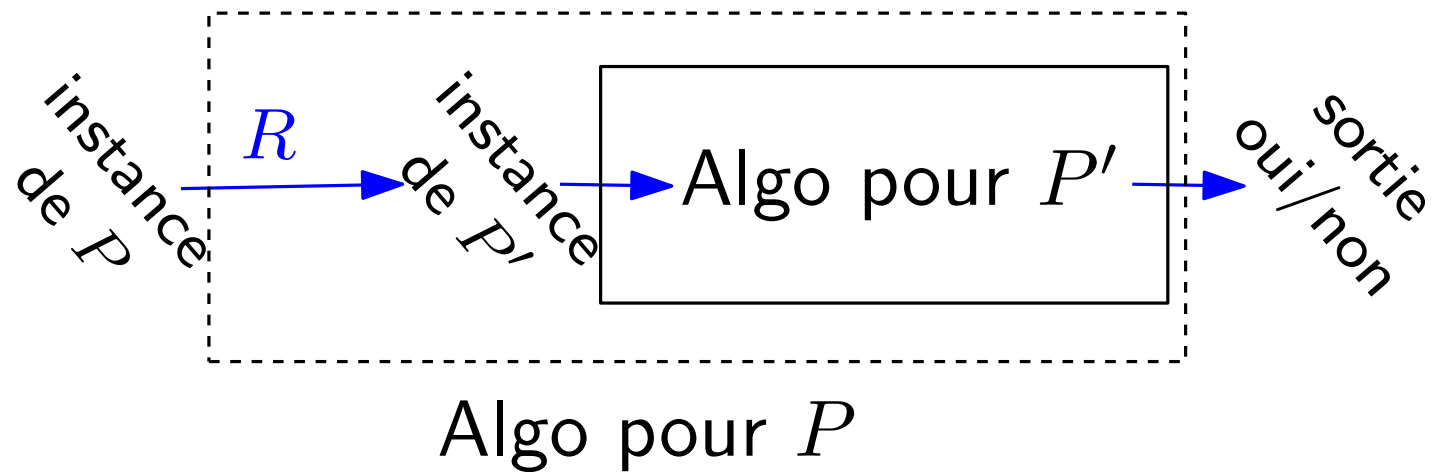
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

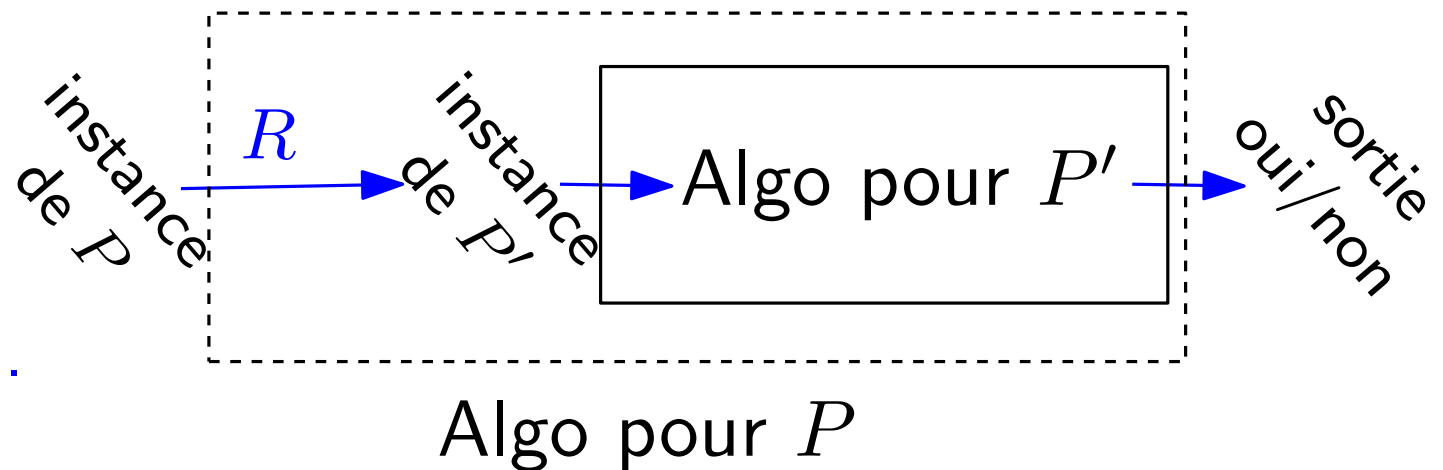


Si on trouve un algo polynomial pour P' , on en a un aussi pour P .

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

Si P se réduit à P' ,
 P plus facile que P' .

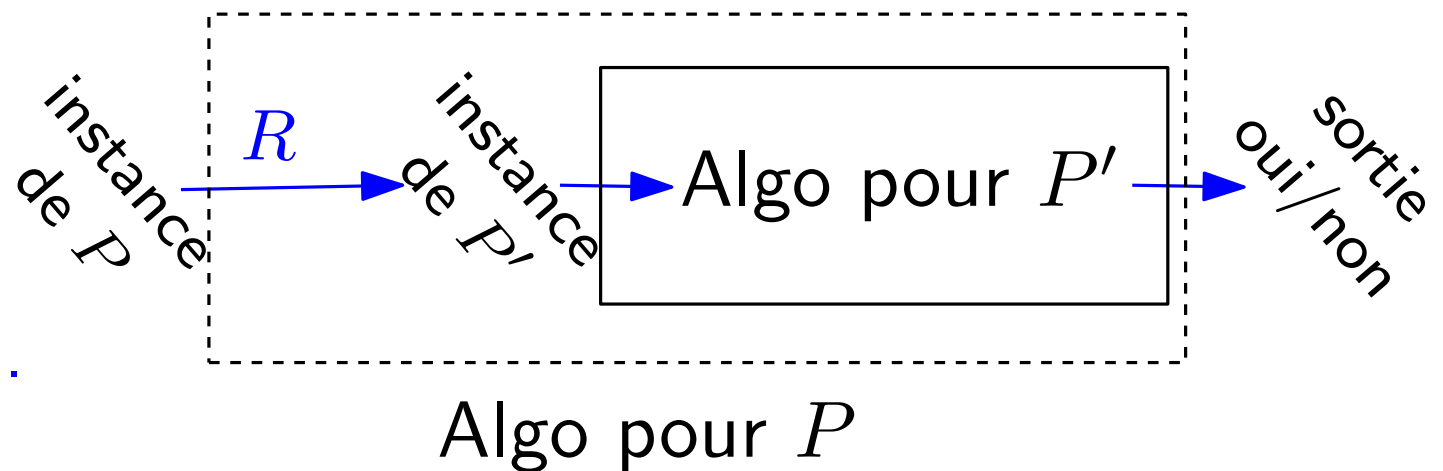


Si on trouve un algo polynomial pour P' , on en a un aussi pour P .

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

Si P se réduit à P' ,
 P plus facile que P' .



Si on trouve un algo polynomial pour P' , on en a un aussi pour P .

Cette notion de réduction est restrictive: un seul appel à P' à la fin, on ne change pas la réponse ("many-one polynomial time reduction").

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure ReduitVariables($\sum_{j=1}^n A_{i,j}x_j \leq b_i, i = 1, \dots, m$):

- $J := \{1, \dots, n\}$
- Pour $k := 1, \dots, n$ faire
 - si Faisable($\sum_{j \in J \setminus \{k\}} A_{i,j}x_j \leq b_i, i = 1, \dots, m$)
 - alors $J := J \setminus \{k\}$.

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure ReduitVariables($\sum_{j=1}^n A_{i,j}x_j \leq b_i, i = 1, \dots, m$):

- $J := \{1, \dots, n\}$
- Pour $k := 1, \dots, n$ faire
 - si Faisable($\sum_{j \in J \setminus \{k\}} A_{i,j}x_j \leq b_i, i = 1, \dots, m$)
 - alors $J := J \setminus \{k\}$. ($x_k = 0$)

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitVariables`($\sum_{j=1}^n A_{i,j}x_j \leq b_i, i = 1, \dots, m$):

- $J := \{1, \dots, n\}$
- Pour $k := 1, \dots, n$ faire
 - si Faisable($\sum_{j \in J \setminus \{k\}} A_{i,j}x_j \leq b_i, i = 1, \dots, m$)
 - alors $J := J \setminus \{k\}$. ($x_k = 0$)

Remarques: Si $S' = \text{ReduitVariables}(S)$ est faisable, alors S aussi.

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitVariables`($\sum_{j=1}^n A_{i,j}x_j \leq b_i, i = 1, \dots, m$):

- $J := \{1, \dots, n\}$
- Pour $k := 1, \dots, n$ faire
 - si Faisable($\sum_{j \in J \setminus \{k\}} A_{i,j}x_j \leq b_i, i = 1, \dots, m$)
 - alors $J := J \setminus \{k\}$. ($x_k = 0$)

Remarques: Si $S' = \text{ReduitVariables}(S)$ est faisable, alors S aussi.

Les solutions de S' sont à coordonnées non nulles.

Réductions entre problèmes: réduction de Turing

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitVariables`($\sum_{j=1}^n A_{i,j}x_j \leq b_i, i = 1, \dots, m$):

- $J := \{1, \dots, n\}$
- Pour $k := 1, \dots, n$ faire
 - si Faisable($\sum_{j \in J \setminus \{k\}} A_{i,j}x_j \leq b_i, i = 1, \dots, m$)
 - alors $J := J \setminus \{k\}$. ($x_k = 0$)

Remarques: Si $S' = \text{ReduitVariables}(S)$ est faisable, alors S aussi.

Les solutions de S' sont à coordonnées non nulles.

Le calcul de S' se fait en temps polynomial.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si faisable $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, \quad i \in I \cup \{\ell\} \end{array} \right)$
- Alors $I := I \cup \{\ell\}$.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si faisable $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, \quad i \in I \cup \{\ell\} \end{array} \right)$

- Alors $I := I \cup \{\ell\}$.

Remarques: Par construction on obtient un système faisable avec

$$(1) \quad \sum_{j \in J} A_{ij} x_j = b_i, \quad \text{pour tout } i \in I.$$

Si ce système est de rang plein on a construit une solution à S' et S .

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si faisable $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, \quad i \in I \cup \{\ell\} \end{array} \right)$

- Alors $I := I \cup \{\ell\}$.

Remarques: Par construction on obtient un système faisable avec

$$(1) \quad \sum_{j \in J} A_{ij} x_j = b_i, \quad \text{pour tout } i \in I.$$

Si ce système est de rang plein on a construit une solution à S' et S .

Sinon, on a 2 sols $x \neq x'$ de S' et $\lambda x + \mu x'$ satisfait (1), $\forall \lambda + \mu = 1$.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si faisable $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, \quad i \in I \cup \{\ell\} \end{array} \right)$

- Alors $I := I \cup \{\ell\}$.

Remarques: Par construction on obtient un système faisable avec

$$(1) \quad \sum_{j \in J} A_{ij} x_j = b_i, \quad \text{pour tout } i \in I.$$

Si ce système est de rang plein on a construit une solution à S' et S .

Sinon, on a 2 sols $x \neq x'$ de S' et $\lambda x + \mu x'$ satisfait (1), $\forall \lambda + \mu = 1$.

soit on peut annuler une coordonnée, soit on rencontre une inégalité.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme faisable(S) qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si faisable $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, i \in I \cup \{\ell\} \end{array} \right)$
- Alors $I := I \cup \{\ell\}$.

Remarques: On a utilisé plusieurs fois l'algorithme faisable comme une boîte noire (on parle aussi d'oracle).

Les réductions de ce type sont appelées réduction au sens de Turing.

Réductions entre problèmes: deuxième exemple

Donnée: Un algorithme `faisable(S)` qui décide de l'existence d'une solution à un système d'inéquations linéaires en temps polynomial.

Problème: Trouver une solution d'un système d'inéquations linéaires.

Procédure `ReduitEquations(S')`

- $I := \emptyset$

- Pour $\ell := 1, \dots, m$ faire

- Si `faisable` $\left(\begin{array}{l} \sum_{j \in J} A_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\}, \\ \sum_{j \in J} A_{ij} x_j \geq b_i, \quad i \in I \cup \{\ell\} \end{array} \right)$

- Alors $I := I \cup \{\ell\}$.

Remarques: On a utilisé plusieurs fois l'algorithme `faisable` comme une boîte noire (on parle aussi d'oracle).

Les réductions de ce type sont appelées réduction au sens de Turing.

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

Problèmes de décision, classe \mathcal{P}

L'ensemble des problèmes de décision qui admettent un algorithme de **résolution en temps polynomial** forme la **classe \mathcal{P}** .

Exemples:

Arbre couvrant de poids $\leq k$ est dans \mathcal{P} : Kruskal ou Prim

Flot entier de valeur $\leq k$ est dans \mathcal{P} : Ford Fulkerson

Programmation linéaire dans \mathbb{R}^d est dans \mathcal{P} : méthode de l'ellipsoïde

Programmation linéaire dans \mathbb{Z}^d , **on ne sait pas s'il est dans \mathcal{P}** .

Il existe des problèmes dont on sait qu'ils ne sont pas dans \mathcal{P} : arrêt d'une machine de Turing en k étapes, max cut sur des graphes représenté par circuits compacts...

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un **certificat** qui peut être **vérifié** par un algorithme en temps polynomial.

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un certificat qui peut être vérifié par un algorithme en temps polynomial.

En particulier le certificat doit être de taille polynomiale.

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un certificat qui peut être vérifié par un algorithme en temps polynomial.

En particulier le certificat doit être de taille polynomiale.

Souvent le certificat est en même temps la réponse au problème de recherche associé à la décision:

Exemple. Existe-t-il une solution à un système d'inéquation linéaire ?

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un certificat qui peut être vérifié par un algorithme en temps polynomial.

En particulier le certificat doit être de taille polynomiale.

Souvent le certificat est en même temps la réponse au problème de recherche associé à la décision:

Exemple. Existe-t-il une solution à un système d'inéquation linéaire ?

La donnée d'un point satisfaisant les inéquations est un certificat: on le vérifie qu'il est correct en temps polynomial

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un **certificat** qui peut être vérifié par un algorithme en temps polynomial.

Reformulation: $D \in \mathcal{NP}$ s'il existe $D' \in \mathcal{P}$, tel que pour toute instance w de D : $w \in \text{Oui}(D)$ ssi il existe w' tq $(w, w') \in \text{Oui}(D')$.

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un certificat qui peut être vérifié par un algorithme en temps polynomial.

Reformulation: $D \in \mathcal{NP}$ s'il existe $D' \in \mathcal{P}$, tel que pour toute instance w de D : $w \in \text{Oui}(D)$ ssi il existe w' tq $(w, w') \in \text{Oui}(D')$.

En particulier $\mathcal{P} \subset \mathcal{NP}$: pas besoin de certificat pour vérifier.

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un **certificat** qui peut être vérifié par un algorithme en temps polynomial.

Reformulation: $D \in \mathcal{NP}$ s'il existe $D' \in \mathcal{P}$, tel que pour toute instance w de D : $w \in \text{Oui}(D)$ ssi il existe w' tq $(w, w') \in \text{Oui}(D')$.

En particulier $\mathcal{P} \subset \mathcal{NP}$: pas besoin de certificat pour vérifier.

Reformulation: $D \in \mathcal{NP}$ s'il existe un algorithme **non déterministe** (avec des instructions de branchement non déterministe) tel que

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un **certificat** qui peut être vérifié par un algorithme en temps polynomial.

Reformulation: $D \in \mathcal{NP}$ s'il existe $D' \in \mathcal{P}$, tel que pour toute instance w de D : $w \in \text{Oui}(D)$ ssi il existe w' tq $(w, w') \in \text{Oui}(D')$.

En particulier $\mathcal{P} \subset \mathcal{NP}$: pas besoin de certificat pour vérifier.

Reformulation: $D \in \mathcal{NP}$ s'il existe un algorithme **non déterministe** (avec des instructions de branchement non déterministe) tel que

- pour chaque instance $w \in \text{Oui}(D)$ il existe une exécution de l'algorithme sur w qui répond **Oui** en temps polynomial

La classe \mathcal{NP} : *Non-determinist Polynomial*

La classe \mathcal{NP} est formée des problèmes de décision D dont une instance est dans $\text{Oui}(D)$ ssi elle admet un **certificat** qui peut être vérifié par un algorithme en temps polynomial.

Reformulation: $D \in \mathcal{NP}$ s'il existe $D' \in \mathcal{P}$, tel que pour toute instance w de D : $w \in \text{Oui}(D)$ ssi il existe w' tq $(w, w') \in \text{Oui}(D')$.

En particulier $\mathcal{P} \subset \mathcal{NP}$: pas besoin de certificat pour vérifier.

Reformulation: $D \in \mathcal{NP}$ s'il existe un algorithme **non déterministe** (avec des instructions de branchement non déterministe) tel que

- pour chaque instance $w \in \text{Oui}(D)$ il existe une exécution de l'algorithme sur w qui répond **Oui** en temps polynomial
- pour toute instance $w \notin \text{Oui}(D)$ il n'existe aucune exécution de l'algorithme sur w qui réponde **Oui**

Exemple de problème dans la classe \mathcal{NP} .

Notre problème \mathcal{D} : le **problème des chemins hamiltoniens**.

Donnée: un graphe G non orienté.

Problème: existe-t-il un chemin qui passe une fois et une seule par chaque sommet ?

Exemple de problème dans la classe \mathcal{NP} .

Notre problème \mathcal{D} : le **problème des chemins hamiltoniens**.

Donnée: un graphe G non orienté.

Problème: existe-t-il un chemin qui passe une fois et une seule par chaque sommet ?

Le problème \mathcal{D}' associé:

Donnée: un graphe G non orienté, une suite v de sommets

Problème: la suite v est elle un chemin hamiltonien de G ?

Exemple de problème dans la classe \mathcal{NP} .

Notre problème \mathcal{D} : le **problème des chemins hamiltoniens**.

Donnée: un graphe G non orienté.

Problème: existe-t-il un chemin qui passe une fois et une seule par chaque sommet ?

Le problème \mathcal{D}' associé:

Donnée: un graphe G non orienté, une suite v de sommets

Problème: la suite v est elle un chemin hamiltonien de G ?

Évidemment \mathcal{D}' est dans \mathcal{P} , donc \mathcal{D} est dans \mathcal{NP} .

Exemple de problème dans la classe \mathcal{NP} .

Notre problème \mathcal{D} : le **problème des chemins hamiltoniens**.

Donnée: un graphe G non orienté.

Problème: existe-t-il un chemin qui passe une fois et une seule par chaque sommet ?

Le problème \mathcal{D}' associé:

Donnée: un graphe G non orienté, une suite v de sommets

Problème: la suite v est elle un chemin hamiltonien de G ?

Évidemment \mathcal{D}' est dans \mathcal{P} , donc \mathcal{D} est dans \mathcal{NP} .

Autrement dit, il existe un chemin hamiltonien dans G si et seulement il existe v tel que v soit un chemin hamiltonien de G .

Exemple de problème dans la classe \mathcal{NP} .

Notre problème \mathcal{D} : le **problème des chemins hamiltoniens**.

Donnée: un graphe G non orienté.

Problème: existe-t-il un chemin qui passe une fois et une seule par chaque sommet ?

Le problème \mathcal{D}' associé:

Donnée: un graphe G non orienté, une suite v de sommets

Problème: la suite v est elle un chemin hamiltonien de G ?

Évidemment \mathcal{D}' est dans \mathcal{P} , donc \mathcal{D} est dans \mathcal{NP} .

Autrement dit, il existe un chemin hamiltonien dans G si et seulement il existe v tel que v soit un chemin hamiltonien de G .
Devoir décider l'existence d'un objet ayant une propriété facile à vérifier en temps polynomial conduit immédiatement à \mathcal{NP} .

Problèmes de décision, classe \mathcal{P} , classe \mathcal{NP}

La conjecture de l'informatique

$$\mathcal{P} \stackrel{?}{\neq} \mathcal{NP}$$

Problèmes de décision, classe \mathcal{P} , classe \mathcal{NP}

La conjecture de l'informatique

$$\mathcal{P} \stackrel{?}{\neq} \mathcal{NP}$$

On ne sait pas prouver que:

- appeler un oracle qui donne la solution + vérifier en temps polynomial est une méthode plus puissante que
- trouver soi-même la solution en temps polynomial

Problèmes de décision, classe \mathcal{P} , classe \mathcal{NP}

La conjecture de l'informatique

$$\mathcal{P} \stackrel{?}{\neq} \mathcal{NP}$$

On ne sait pas prouver que:

- appeler un oracle qui donne la solution + vérifier en temps polynomial est une méthode plus puissante que
- trouver soi-même la solution en temps polynomial

Cette question théorique a des implications très concrètes: par exemple à la sûreté de nombreuses procédures cryptographiques.

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

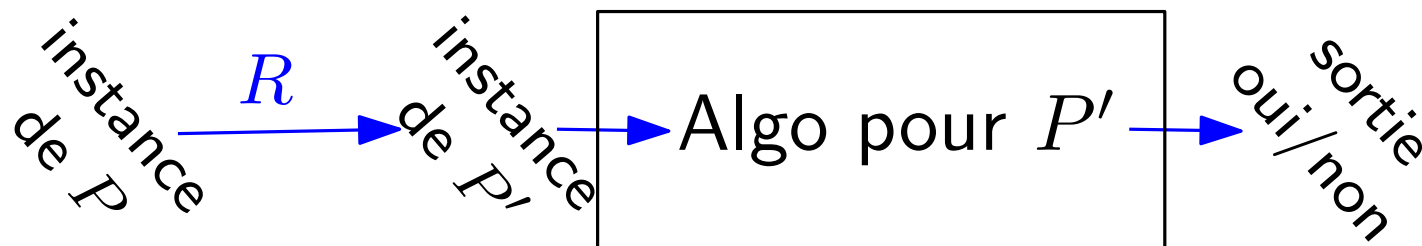
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



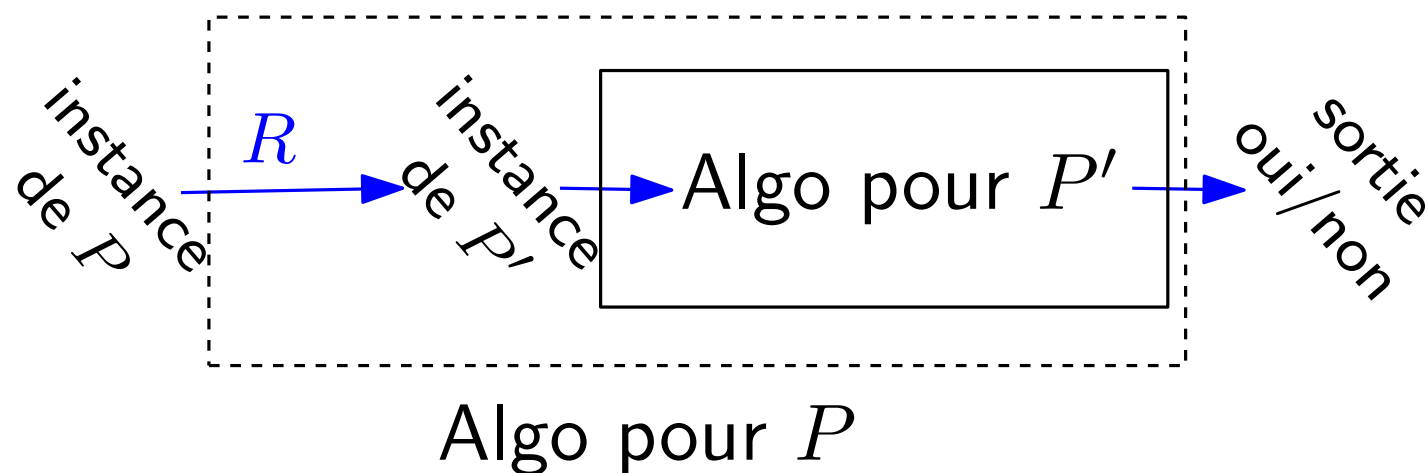
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



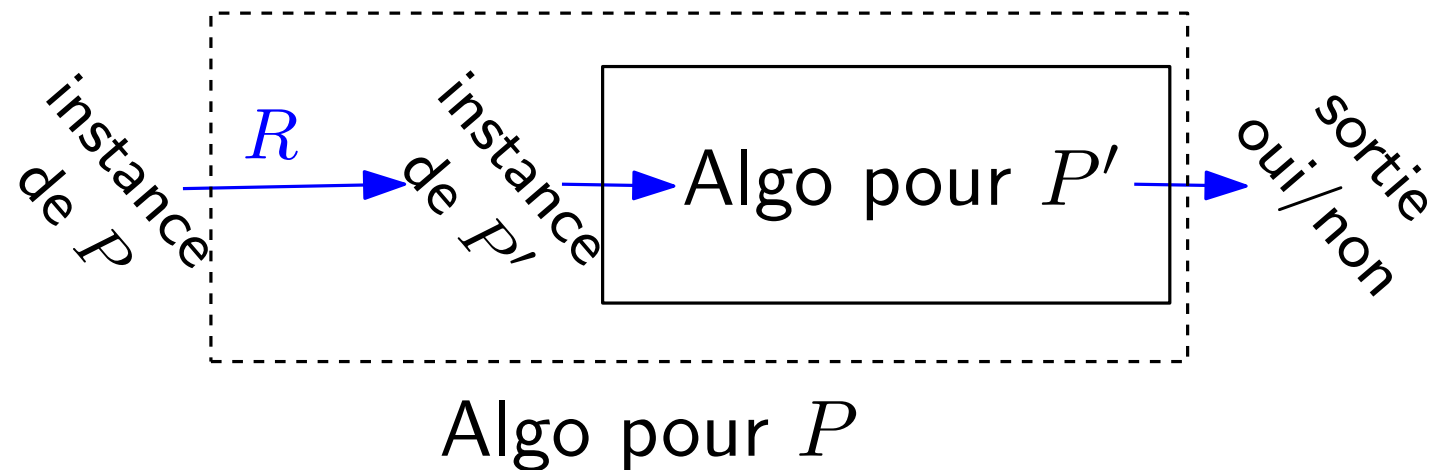
Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



Réduction polynomiale d'un problème à un autre

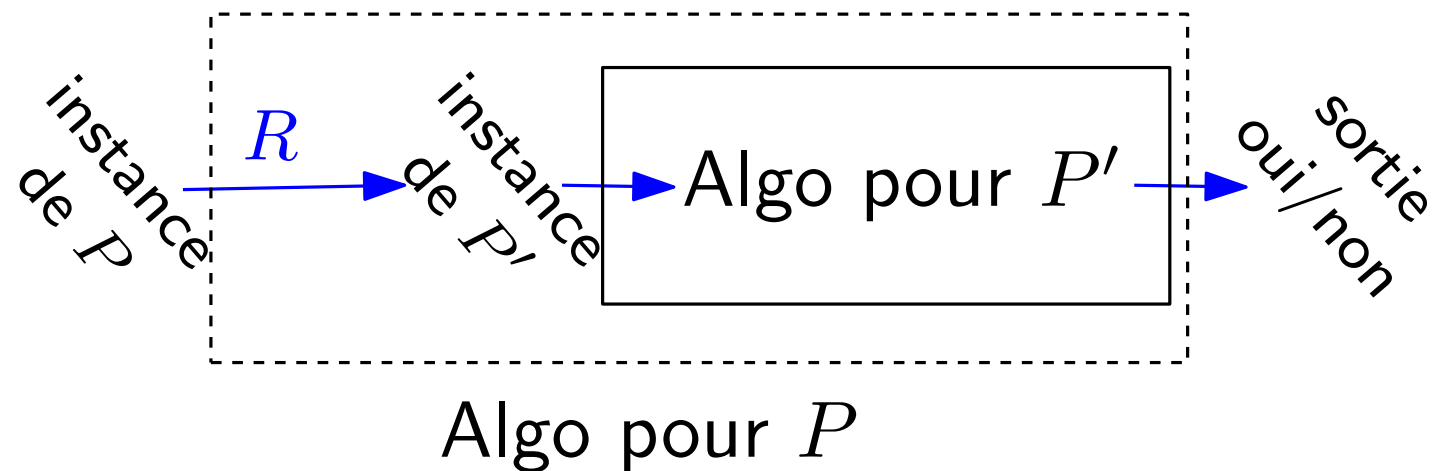
Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.



Si on connaît vraiment un algo polynomial pour P' , alors $P \in \mathcal{P}$.

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

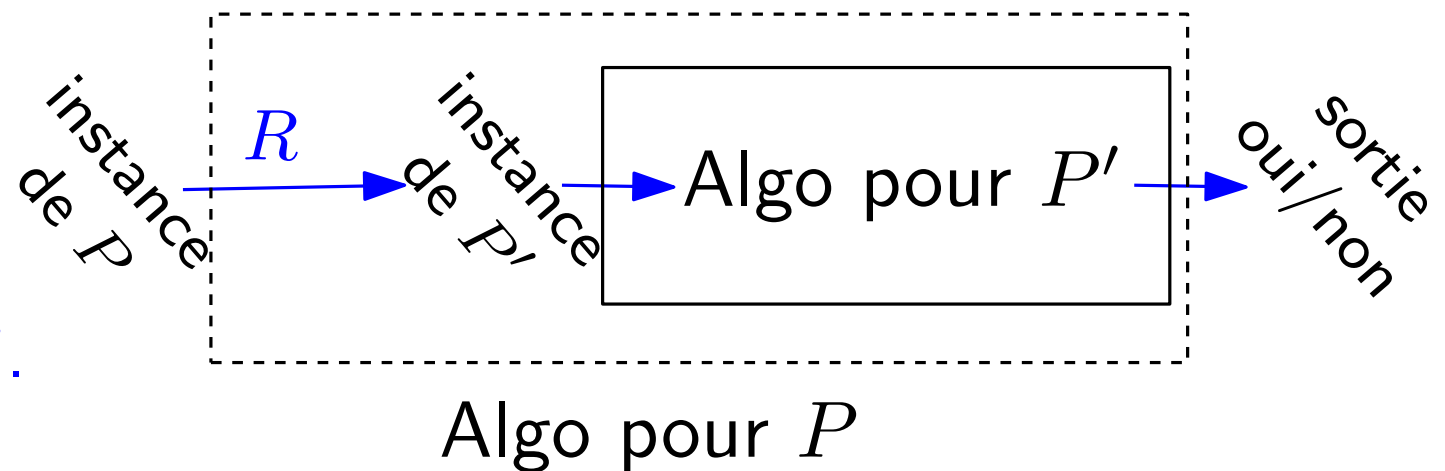


Si on connaît vraiment un algo polynomial pour P' , alors $P \in \mathcal{P}$.
Si on savait que P est dur ($P \notin \mathcal{P}$) alors P' aussi ($P' \notin \mathcal{P}$).

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

Si P se réduit à P' ,
 P plus facile que P' .

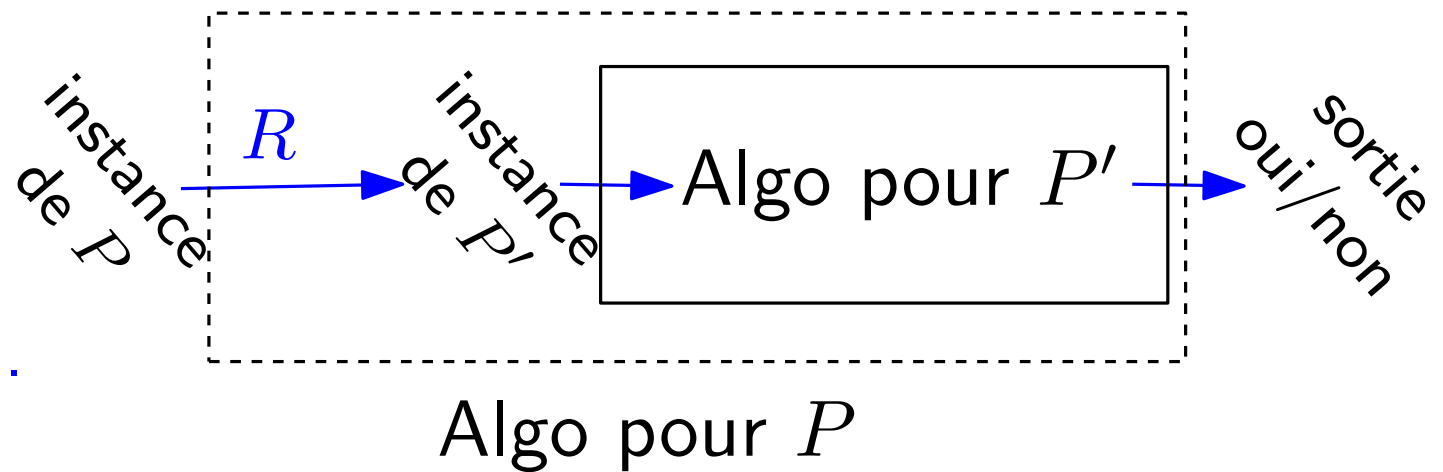


Si on connaît vraiment un algo polynomial pour P' , alors $P \in \mathcal{P}$.
Si on savait que P est dur ($P \notin \mathcal{P}$) alors P' aussi ($P' \notin \mathcal{P}$).

Réduction polynomiale d'un problème à un autre

Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme **polynomial** qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \text{Oui}(P) \Leftrightarrow u' \in \text{Oui}(P')$.

Si P se réduit à P' ,
 P plus facile que P' .



Si on connaît vraiment un algo polynomial pour P' , alors $P \in \mathcal{P}$.
Si on savait que P est dur ($P \notin \mathcal{P}$) alors P' aussi ($P' \notin \mathcal{P}$).

Transitivité: $P \rightarrow P'$ et $P' \rightarrow P''$ alors $P \rightarrow P''$.

Problème \mathcal{NP} -dur, \mathcal{NP} -complétude

Un problème de décision Q est \mathcal{NP} -dur si pour tout problème P de \mathcal{NP} , il existe une réduction polynomiale de P à Q .

Problème \mathcal{NP} -dur, \mathcal{NP} -complétude

Un **problème de décision** Q est \mathcal{NP} -dur si pour tout problème P de \mathcal{NP} , il existe une réduction polynomiale de P à Q .

Un problème est \mathcal{NP} -complet s'il est dans \mathcal{NP} et \mathcal{NP} -dur.

Problème \mathcal{NP} -dur, \mathcal{NP} -complétude

Un problème de décision Q est \mathcal{NP} -dur si pour tout problème P de \mathcal{NP} , il existe une réduction polynomiale de P à Q .

Un problème est \mathcal{NP} -complet s'il est dans \mathcal{NP} et \mathcal{NP} -dur.

Théorème.

- si P est \mathcal{NP} -dur et $P \in \mathcal{P}$ alors $\mathcal{P} = \mathcal{NP}$.
- si P est \mathcal{NP} -dur et si $\mathcal{P} \neq \mathcal{NP}$ alors il n'existe pas d'algorithme polynomial pour P .

Problème \mathcal{NP} -dur, \mathcal{NP} -complétude

Un problème de décision Q est \mathcal{NP} -dur si pour tout problème P de \mathcal{NP} , il existe une réduction polynomiale de P à Q .

Un problème est \mathcal{NP} -complet s'il est dans \mathcal{NP} et \mathcal{NP} -dur.

Théorème.

- si P est \mathcal{NP} -dur et $P \in \mathcal{P}$ alors $\mathcal{P} = \mathcal{NP}$.
- si P est \mathcal{NP} -dur et si $\mathcal{P} \neq \mathcal{NP}$ alors il n'existe pas d'algorithme polynomial pour P .

ce qu'on croit

En attendant preuve du contraire,

si P est \mathcal{NP} -dur alors P peut être considéré comme dur...

Problème \mathcal{NP} -dur, \mathcal{NP} -complétude

Un problème de décision Q est \mathcal{NP} -dur si pour tout problème P de \mathcal{NP} , il existe une réduction polynomiale de P à Q .

Un problème est \mathcal{NP} -complet s'il est dans \mathcal{NP} et \mathcal{NP} -dur.

Théorème.

- si P est \mathcal{NP} -dur et $P \in \mathcal{P}$ alors $\mathcal{P} = \mathcal{NP}$.
- si P est \mathcal{NP} -dur et si $\mathcal{P} \neq \mathcal{NP}$ alors il n'existe pas d'algorithme polynomial pour P .

ce qu'on croit

En attendant preuve du contraire,

si P est \mathcal{NP} -dur alors P peut être considéré comme dur...

Pour que cette définition ait un intérêt il faudrait qu'il existe un problème \mathcal{NP} -complet, ce qui n'est pas évident...

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

SAT, un premier problème \mathcal{NP} -complet.

Donnée: une formule booléenne en forme normale conjonctive (CNF):
conjonction de disjonction de la forme $(x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge \dots$

Problème: existe-t-il une assignation des variables qui rende la
formule vraie.

SAT, un premier problème \mathcal{NP} -complet.

Donnée: une formule booléenne en forme normale conjonctive (CNF):
conjonction de disjonction de la forme $(x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge \dots$

Problème: existe-t-il une assignation des variables qui rende la formule vraie.

Lemme: $SAT \in \mathcal{NP}$.

une fois devinée la solution, vérifier que la formule est vraie est facile.

SAT, un premier problème \mathcal{NP} -complet.

Donnée: une formule booléenne en forme normale conjonctive (CNF):
conjonction de disjonction de la forme $(x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge \dots$

Problème: existe-t-il une assignation des variables qui rende la formule vraie.

Lemme: $\text{SAT} \in \mathcal{NP}$.

une fois devinée la solution, vérifier que la formule est vraie est facile.

Théorème de Cook: Soit P un problème de \mathcal{NP} .

Il existe un polynôme $p(n)$ et un algorithme qui pour chaque entrée w de P , construit en temps $p(|w|)$ une formule booléenne f_w en CNF telle que:

$$w \in \text{Oui}(P) \quad \Leftrightarrow \quad f_w \text{ satisfaisable}$$

SAT, un premier problème \mathcal{NP} -complet.

Donnée: une formule booléenne en forme normale conjonctive (CNF):
conjonction de disjonction de la forme $(x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge \dots$

Problème: existe-t-il une assignation des variables qui rende la formule vraie.

Lemme: $\text{SAT} \in \mathcal{NP}$.

une fois devinée la solution, vérifier que la formule est vraie est facile.

Théorème de Cook: Soit P un problème de \mathcal{NP} .

Il existe un polynôme $p(n)$ et un algorithme qui pour chaque entrée w de P , construit en temps $p(|w|)$ une formule booléenne f_w en CNF telle que:

$$w \in \text{Oui}(P) \iff f_w \text{ satisfaisable}$$

Résultat admis: la preuve consiste en un codage d'une machine de Turing qui vérifie les solutions de P en temps polynomial.

Réduction et \mathcal{NP} -complétude

Théorème de Cook (reformulation):

Le problème SAT est \mathcal{NP} -complet.

Réduction et \mathcal{NP} -complétude

Théorème de Cook (reformulation):

Le problème SAT est \mathcal{NP} -complet.

On va utiliser ce résultat pour en montrer d'autres par réduction.

Réduction et \mathcal{NP} -complétude

Théorème de Cook (reformulation):

Le problème SAT est \mathcal{NP} -complet.

On va utiliser ce résultat pour en montrer d'autres par réduction.

Principe de réduction: Soit P un problème de décision.

Si le problème SAT se réduit polynomialement à P
alors P est \mathcal{NP} -dur.

Réduction et \mathcal{NP} -complétude

Théorème de Cook (reformulation):

Le problème SAT est \mathcal{NP} -complet.

On va utiliser ce résultat pour en montrer d'autres par réduction.

Principe de réduction: Soit P un problème de décision.

Si le problème SAT se réduit polynomialement à P
alors P est \mathcal{NP} -dur.

Rappel mnémotechnique: Si Q se réduit à P ça veut dire que

P est au moins aussi dur à traiter que Q :

⇒ réduire le problème dont on sait déjà qu'il est dur au ptit nouveau.

Réduction et \mathcal{NP} -complétude

Théorème de Cook (reformulation):

Le problème SAT est \mathcal{NP} -complet.

On va utiliser ce résultat pour en montrer d'autres par réduction.

Principe de réduction: Soit P un problème de décision.

Si le problème SAT se réduit polynomialement à P
alors P est \mathcal{NP} -dur.

Rappel mnémotechnique: Si Q se réduit à P ça veut dire que

P est au moins aussi dur à traiter que Q :

⇒ réduire le problème dont on sait déjà qu'il est dur au ptit nouveau.

On fait tous au moins une fois le contraire dans sa vie...

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

Lemme: 3-SAT est dans \mathcal{NP} .

Preuve. Deviner l'assignation et vérifier que la formule est vraie.

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

Lemme: 3-SAT est dans \mathcal{NP} .

Preuve. Deviner l'assignation et vérifier que la formule est vraie.

SAT se réduit à 3-SAT: Étant donné une instance du problème SAT (une formule), il faut trouver une instance de 3-SAT équivalente.

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

Lemme: 3-SAT est dans \mathcal{NP} .

Preuve. Deviner l'assignation et vérifier que la formule est vraie.

SAT se réduit à 3-SAT: Étant donné une instance du problème SAT (une formule), il faut trouver une instance de 3-SAT équivalente.

On remplace chaque clause $C = x \vee y \vee z \vee t \vee u \vee \dots$ par une conjonction de clause à 3 littéraux *connectées* par de nouvelles variables:

$$R(C) = (x \vee y \vee a) \wedge (\bar{a} \vee z \vee b) \wedge (\bar{b} \vee t \vee c) \wedge (\bar{c} \vee u \vee d) \wedge \dots$$

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

Lemme: 3-SAT est dans \mathcal{NP} .

Preuve. Deviner l'assignation et vérifier que la formule est vraie.

SAT se réduit à 3-SAT: Étant donné une instance du problème SAT (une formule), il faut trouver une instance de 3-SAT équivalente.

On remplace chaque clause $C = x \vee y \vee z \vee t \vee u \vee \dots$ par une conjonction de clause à 3 littéraux *connectées* par de nouvelles variables:

$$R(C) = (x \vee y \vee a) \wedge (\bar{a} \vee z \vee b) \wedge (\bar{b} \vee t \vee c) \wedge (\bar{c} \vee u \vee d) \wedge \dots$$

La formule $F = C_1 \wedge C_2 \wedge \dots$ est satisfaisable si et seulement si la formule $R(F) = R(C_1) \wedge R(C_2) \wedge \dots$ l'est. On construit $R(F)$ en temps polynomial.

3-SAT est \mathcal{NP} -complet. (Remarque: $2\text{-SAT} \in \mathcal{P}$)

3-SAT: les clauses contiennent au plus 3 littéraux.

$$(x_{i_1} \vee x_{j_1} \vee x_{k_1}) \wedge (x_{i_2} \vee x_{j_2} \vee x_{j_3}) \wedge \dots$$

Lemme: 3-SAT est dans \mathcal{NP} .

Preuve. Deviner l'assignation et vérifier que la formule est vraie.

SAT se réduit à 3-SAT: Étant donné une instance du problème SAT (une formule), il faut trouver une instance de 3-SAT équivalente.

On remplace chaque clause $C = x \vee y \vee z \vee t \vee u \vee \dots$ par une conjonction de clause à 3 littéraux *connectées* par de nouvelles variables:

$$R(C) = (x \vee y \vee a) \wedge (\bar{a} \vee z \vee b) \wedge (\bar{b} \vee t \vee c) \wedge (\bar{c} \vee u \vee d) \wedge \dots$$

La formule $F = C_1 \wedge C_2 \wedge \dots$ est satisfaisable si et seulement si la formule $R(F) = R(C_1) \wedge R(C_2) \wedge \dots$ l'est. On construit $R(F)$ en temps polynomial.

CQFD

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

STABLE est \mathcal{NP} -complet.

Le problème STABLE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets, ne contenant aucune paire de sommets voisins ?

On appelle un tel ensemble un **stable** ou un **ensemble indépendant** de G .

STABLE est \mathcal{NP} -complet.

Le problème STABLE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets, ne contenant aucune paire de sommets voisins ?

On appelle un tel ensemble un **stable** ou un **ensemble indépendant** de G .

3-SAT se réduit à STABLE: étant donnée une instance de 3-SAT, c-à-d une formule f avec n clauses à 3 littéraux, chercher une instance $R(f) = (G, k)$ de STABLE tq G contient un stable de taille k ssi f est satisfaisable.

STABLE est \mathcal{NP} -complet.

Le problème STABLE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets, ne contenant aucune paire de sommets voisins ?

On appelle un tel ensemble un **stable** ou un **ensemble indépendant** de G .

3-SAT se réduit à STABLE: étant donnée une instance de 3-SAT, c-à-d une formule f avec n clauses à 3 littéraux, chercher une instance $R(f) = (G, k)$ de STABLE tq G contient un stable de taille k ssi f est satisfaisable.

On construit un graphe G avec $3n$ sommets, un pour chaque occurrence de variable dans une clause, reliés en triangles associés à chaque clause et avec des arêtes supplémentaires entre toutes les occurrences de x et toutes les occurrences de \bar{x} pour chaque variable x .

STABLE est \mathcal{NP} -complet.

Le problème STABLE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets, ne contenant aucune paire de sommets voisins ?

On appelle un tel ensemble un **stable** ou un **ensemble indépendant** de G .

3-SAT se réduit à STABLE: étant donnée une instance de 3-SAT, c-à-d une formule f avec n clauses à 3 littéraux, chercher une instance $R(f) = (G, k)$ de STABLE tq G contient un stable de taille k ssi f est satisfaisable.

On construit un graphe G avec $3n$ sommets, un pour chaque occurrence de variable dans une clause, reliés en triangles associés à chaque clause et avec des arêtes supplémentaires entre toutes les occurrences de x et toutes les occurrences de \bar{x} pour chaque variable x .

La formule f est satisfaisable si et seulement si G a un stable de taille n .

STABLE est \mathcal{NP} -complet.

Le problème STABLE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets, ne contenant aucune paire de sommets voisins ?

On appelle un tel ensemble un **stable** ou un **ensemble indépendant** de G .

3-SAT se réduit à STABLE: étant donnée une instance de 3-SAT, c-à-d une formule f avec n clauses à 3 littéraux, chercher une instance $R(f) = (G, k)$ de STABLE tq G contient un stable de taille k ssi f est satisfaisable.

On construit un graphe G avec $3n$ sommets, un pour chaque occurrence de variable dans une clause, reliés en triangles associés à chaque clause et avec des arêtes supplémentaires entre toutes les occurrences de x et toutes les occurrences de \bar{x} pour chaque variable x .

La formule f est satisfaisable si et seulement si G a un stable de taille n .
La construction de (G, n) se fait en temps polynomial à partir de f .

CLIQUE et COUVRANT sont \mathcal{NP} -complets.

Le problème CLIQUE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets reliés 2 à 2 ?
On appelle un tel ensemble une **clique**. Clairement dans \mathcal{NP} .

CLIQUE et COUVRANT sont \mathcal{NP} -complets.

Le problème CLIQUE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets reliés 2 à 2 ?
On appelle un tel ensemble une **clique**. Clairement dans \mathcal{NP} .

Le problème CLIQUE est équivalent au problème STABLE dans le graphe complémentaire: étant donné une instance (G, k) de stable, on considère le graphe G^c ayant une arête entre x et y ssi G n'en a pas; G^c a une clique de taille au moins k ssi G a un stable de taille au moins k .

CLIQUE et COUVRANT sont \mathcal{NP} -complets.

Le problème CLIQUE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets reliés 2 à 2 ?
On appelle un tel ensemble une **clique**. Clairement dans \mathcal{NP} .

Le problème CLIQUE est équivalent au problème STABLE dans le graphe complémentaire: étant donné une instance (G, k) de stable, on considère le graphe G^c ayant une arête entre x et y ssi G n'en a pas; G^c a une clique de taille au moins k ssi G a un stable de taille au moins k .

Le problème COUVRANT (=Recouvrement par sommets)

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au plus k sommets couvrants toutes les arêtes (au moins une des 2 extrémités est couverte) ?

CLIQUE et COUVRANT sont \mathcal{NP} -complets.

Le problème CLIQUE:

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au moins k sommets reliés 2 à 2 ?
On appelle un tel ensemble une **clique**. Clairement dans \mathcal{NP} .

Le problème CLIQUE est équivalent au problème STABLE dans le graphe complémentaire: étant donné une instance (G, k) de stable, on considère le graphe G^c ayant une arête entre x et y ssi G n'en a pas; G^c a une clique de taille au moins k ssi G a un stable de taille au moins k .

Le problème COUVRANT (=Recouvrement par sommets)

Donnée: un graphe non orienté G et un entier k .

Problème: Existe-t-il un ensemble d'au plus k sommets couvrants toutes les arêtes (au moins une des 2 extrémités est couverte) ?

Le problème COUVRANT est équivalent au problème STABLE par passage au complémentaire: si $G = (V, E)$ admet un stable X de taille k alors $V \setminus X$ est un couvrant de taille $n - k$ de G .

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P, NP et NP-complétude
- Exemples de problèmes booléens NP-complets.
- Exemples de problèmes de graphes NP-complets
- Exemples de problèmes en nombres entiers NP-complets

SOMME-PARTIELLE est \mathcal{NP} -complet.

Le problème SUBSETSUM:

Donnée: un ensemble fini d'entiers S et un entier s

Problème: Existe-t-il $X \subset S$ tq $\sum_{i \in X} i = s$.

Clairement dans \mathcal{NP} .

SOMME-PARTIELLE est \mathcal{NP} -complet.

Le problème SUBSETSUM:

Donnée: un ensemble fini d'entiers S et un entier s

Problème: Existe-t-il $X \subset S$ tq $\sum_{i \in X} i = s$.

Clairement dans \mathcal{NP} .

COUVRANT se réduit à SOMME-PARTIELLE:

Soit $G = (V, E)$ et k une instance de COUVRANT avec $V = \{x_1, \dots, x_n\}$ et $E = \{e_0, \dots, e_{m-1}\}$. On veut construire S et s tels qu'il existe X avec $\sum_{i \in X} i = s$ ssi G a un ensemble couvrant avec au plus k sommets:

SOMME-PARTIELLE est \mathcal{NP} -complet.

Le problème SUBSETSUM:

Donnée: un ensemble fini d'entiers S et un entier s

Problème: Existe-t-il $X \subset S$ tq $\sum_{i \in X} i = s$.

Clairement dans \mathcal{NP} .

COUVRANT se réduit à SOMME-PARTIELLE:

Soit $G = (V, E)$ et k une instance de COUVRANT avec $V = \{x_1, \dots, x_n\}$ et $E = \{e_0, \dots, e_{m-1}\}$. On veut construire S et s tels qu'il existe X avec $\sum_{i \in X} i = s$ ssi G a un ensemble couvrant avec au plus k sommets:

Au sommet x_i incident aux arêtes e_{i_1}, \dots, e_{i_p} on associe le nombre

$$a_i = \sum_{j=1}^p 4^{i_j} + 4^M, \quad \text{avec } M \text{ assez grand.}$$

On pose $S = \{a_1, \dots, a_n\} \cup \{1, 4, 4^2, \dots, 4^{m-1}\}$, $s = 2(\sum_{j=0}^{m-1} 4^j) + k4^M$.

SOMME-PARTIELLE est \mathcal{NP} -complet.

Le problème SUBSETSUM:

Donnée: un ensemble fini d'entiers S et un entier s

Problème: Existe-t-il $X \subset S$ tq $\sum_{i \in X} i = s$.

Clairement dans \mathcal{NP} .

COUVRANT se réduit à SOMME-PARTIELLE:

Soit $G = (V, E)$ et k une instance de COUVRANT avec $V = \{x_1, \dots, x_n\}$ et $E = \{e_0, \dots, e_{m-1}\}$. On veut construire S et s tels qu'il existe X avec $\sum_{i \in X} i = s$ ssi G a un ensemble couvrant avec au plus k sommets:

Au sommet x_i incident aux arêtes e_{i_1}, \dots, e_{i_p} on associe le nombre

$$a_i = \sum_{j=1}^p 4^{i_j} + 4^M, \quad \text{avec } M \text{ assez grand.}$$

On pose $S = \{a_1, \dots, a_n\} \cup \{1, 4, 4^2, \dots, 4^{m-1}\}$, $s = 2(\sum_{j=0}^{m-1} 4^j) + k4^M$.

Observation. Pour tout choix de $X \subset V$ et $Y \subset \{0, \dots, m-1\}$, la somme associée est de la forme $\sum_{j=0}^{m-1} u_j 4^j + v 4^M$ avec $u_j \in \{0, 1, 2, 3\}$ et $v = |X|$.

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Remarques • On peut le voir comme un problème de programmation linéaire en entier: $\max(\sum a_i x_i \mid \sum p_i x_i \leq P)$.

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Remarques • On peut le voir comme un problème de programmation linéaire en entier: $\max(\sum a_i x_i \mid \sum p_i x_i \leq P)$.

• Un problème de décision naturellement associé: existe-t-il un remplissage de poids au plus P et de gain au moins A ?

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Remarques • On peut le voir comme un problème de programmation linéaire en entier: $\max(\sum a_i x_i \mid \sum p_i x_i \leq P)$.

- Un problème de décision naturellement associé: existe-t-il un remplissage de poids au plus P et de gain au moins A ?
- En particulier pour $a_i = p_i$ et $P = A = s$, c'est SOMME-PARTIELLE.

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Remarques • On peut le voir comme un problème de programmation linéaire en entier: $\max(\sum a_i x_i \mid \sum p_i x_i \leq P)$.

• Un problème de décision naturellement associé: existe-t-il un remplissage de poids au plus P et de gain au moins A ?

• En particulier pour $a_i = p_i$ et $P = A = s$, c'est SOMME-PARTIELLE.

Le sac à dos est donc NP-complet: on ne s'attend donc pas à pouvoir trouver l'optimum en temps polynomial en n (ou plutôt en temps polynomial en $\sum \log p_i + \sum \log a_i + \log P$.)

Optimisation: le sac à dos

Donnée: Les poids p_1, \dots, p_n , les gains a_1, \dots, a_n et la capacité P .

Problème: Trouver $I \subset \{1, \dots, n\}$ tq $\sum_{i \in I} a_i$ soit maximum sous la contrainte $\sum_{i \in I} p_i \leq P$.

Remarques • On peut le voir comme un problème de programmation linéaire en entier: $\max(\sum a_i x_i \mid \sum p_i x_i \leq P)$.

• Un problème de décision naturellement associé: existe-t-il un remplissage de poids au plus P et de gain au moins A ?

• En particulier pour $a_i = p_i$ et $P = A = s$, c'est SOMME-PARTIELLE.

Le sac à dos est donc NP-complet: on ne s'attend donc pas à pouvoir trouver l'optimum en temps polynomial en n (ou plutôt en temps polynomial en $\sum \log p_i + \sum \log a_i + \log P$.)

Cependant le problème est polynomial en n et P (progr. dynamique): on parle de problème NP-complet au sens faible, par opposition aux problèmes NP-complets au sens fort, qui restent NP-complet si les données numériques sont codées en unaire.

Cours 4: Réduction et NP-complétude

- Complexité, problèmes de décision
- Classes P , NP et NP-complétude
- Exemples de problèmes NP-complets.

À retenir: \mathcal{P} , \mathcal{NP} , complétude,
réductions polynomiales (dans le bon sens !)