Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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

Réductions entre problèmes: premiers exemples

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

Problème: Résoudre un problème de programmation linéaire.

Réductions entre problèmes: premiers exemples

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:

Réductions entre problèmes: premiers exemples

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:

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

Réductions entre problèmes: premiers exemples

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:

$$\begin{cases} Ax \leq b \\ \max(c \cdot x) \end{cases} \qquad \begin{cases} Ax \leq b \\ yA \geq c, \; y \geq 0 \\ c \cdot x \geq y \cdot b \end{cases}$$

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.

Réductions entre problèmes: premiers exemples

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: premiers exemples

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: premiers exemples

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

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

Réductions entre problèmes: premiers exemples

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

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

Réductions entre problèmes: premiers exemples

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

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

Remarques: Si $S' =$ `ReduitVariables`(S) est faisable, alors S aussi.

Réductions entre problèmes: premiers exemples

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

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

Remarques: Si $S' =$ `ReduitVariables`(S) est faisable, alors S aussi.

Les solutions de S' sont à coordonnées non nulles.

Réductions entre problèmes: premiers exemples

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

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

Remarques: Si $S' =$ `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: premiers exemples

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: premiers exemples

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 $\ell := 1, \ldots, m$ faire

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

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

Réductions entre problèmes: premiers exemples

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 $\ell := 1, \ldots, m$ faire

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

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

Remarques: Par construction on obtient un système faisable avec

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

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

Réductions entre problèmes: premiers exemples

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 $\ell := 1, \ldots, m$ faire

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

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

Remarques: Par construction on obtient un système faisable avec

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

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: premiers exemples

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 $\ell := 1, \ldots, m$ faire

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

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

Remarques: Par construction on obtient un système faisable avec

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

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.

Complexité

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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 (voir cours INF561)

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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 (voir cours INF561)

On s'intéresse ici à des problèmes décidables, dont on veut mesurer plus finement la complexité.

Un algorithme est de complexité polynomiale s'il existe un polynome 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 $\mathrm{Oui}(D)$ 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 $\mathrm{Oui}(D)$ 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 $= G$; Oui$(D) = \{G$ 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 $\mathrm{Oui}(D)$ 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 $= G$; Oui$(D) = \{G$ 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 $G \setminus \{a\} \in$ 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 $\mathrm{Oui}(D)$ 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 $= G$; Oui$(D) = \{G$ 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 $G \setminus \{a\} \in$ 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
- Existance 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 $\mathrm{Oui}(D)$ 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 $= G$; Oui$(D) = \{G$ 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 $G \setminus \{a\} \in$ 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
- Existance 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

Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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 de décision dont on sait qu'ils ne sont pas dans $\mathcal{P}$ (constructions ad-hoc).

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

La classe $\mathcal{NP}$ est formée des problèmes de décision D dont une instance est dans $\mathrm{Oui}(D)$ 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 $\mathrm{Oui}(D)$ 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 $\mathrm{Oui}(D)$ 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 $\mathrm{Oui}(D)$ 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 $\mathrm{Oui}(D)$ 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: $w \in \mathrm{Oui}(D)$ ssi il existe w' tq $(w, w') \in \mathrm{Oui}(D')$.

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

La classe $\mathcal{NP}$ est formée des problèmes de décision D dont une instance est dans $\mathrm{Oui}(D)$ 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: $w \in \mathrm{Oui}(D)$ ssi il existe w' tq $(w, w') \in \mathrm{Oui}(D')$.

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 $\mathrm{Oui}(D)$ 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: $w \in \mathrm{Oui}(D)$ ssi il existe w' tq $(w, w') \in \mathrm{Oui}(D')$.

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 $\mathrm{Oui}(D)$ 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: $w \in \mathrm{Oui}(D)$ ssi il existe w' tq $(w, w') \in \mathrm{Oui}(D')$.

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 $w \in$ 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 $\mathrm{Oui}(D)$ 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: $w \in \mathrm{Oui}(D)$ ssi il existe w' tq $(w, w') \in \mathrm{Oui}(D')$.

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 $w \in \mathrm{Oui}(D)$ il existe une exécution de l'algorithme sur w qui répond Oui en temps polynomial
- pour toute instance $w \notin \mathrm{Oui}(D)$ 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 théorique

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

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

La conjecture de l'informatique théorique

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

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

Réduction polynomiale d'un problème à un autre

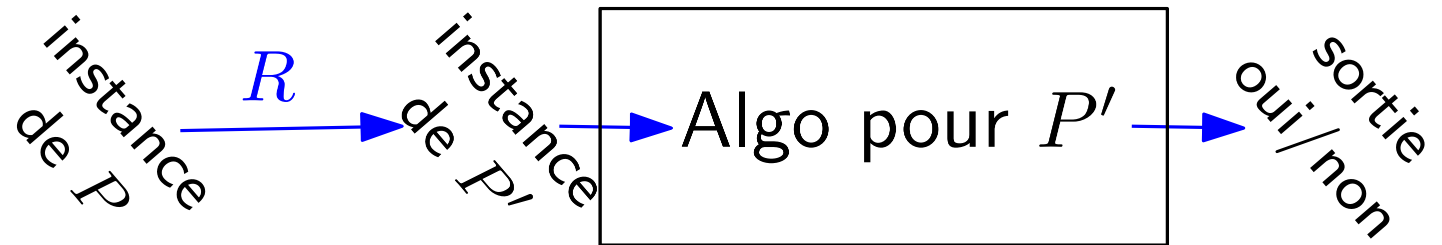
Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme polynomial qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.

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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.

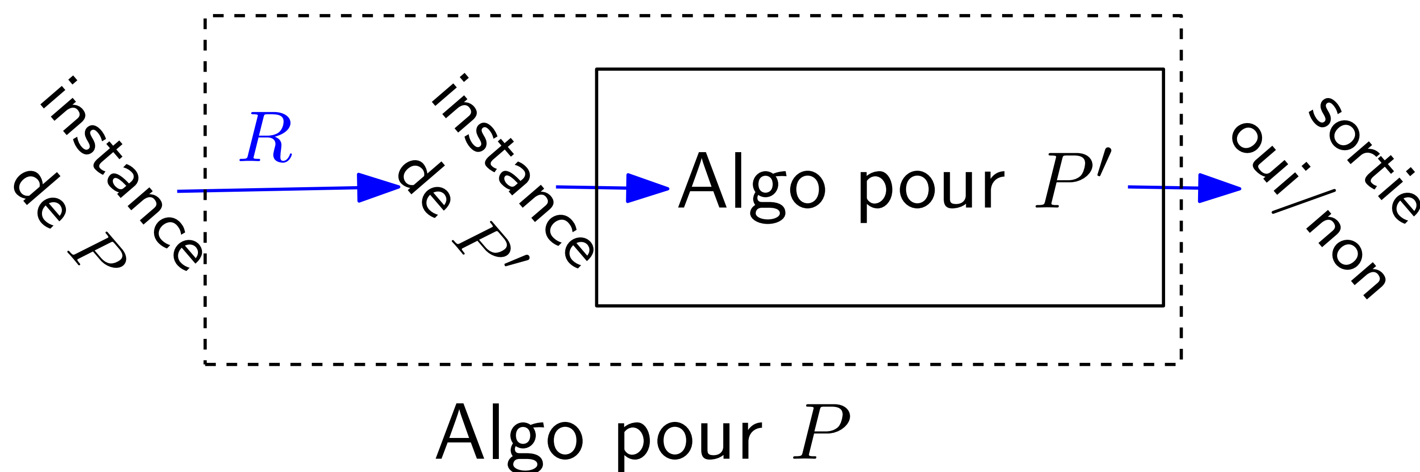
Réduction polynomiale d'un problème à un autre

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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.



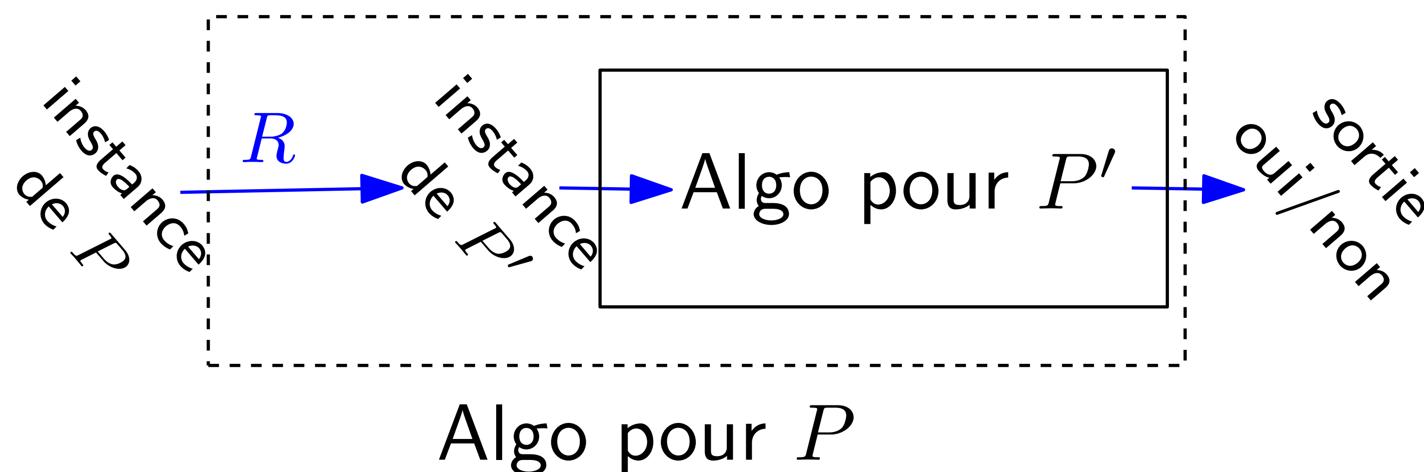
Réduction polynomiale d'un problème à un autre

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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.



Réduction polynomiale d'un problème à un autre

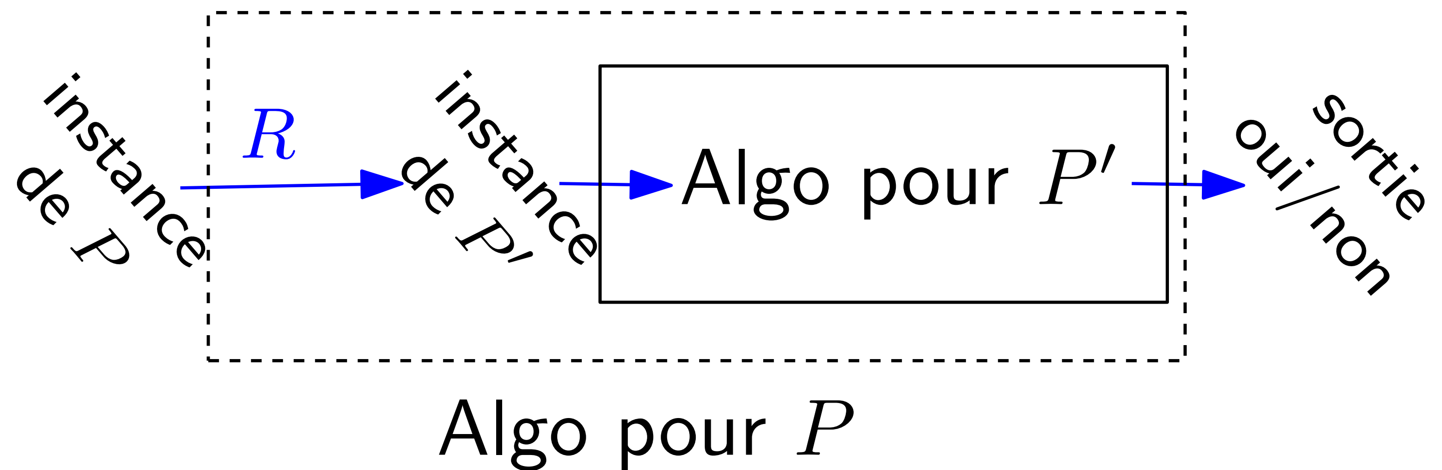
Un problème de décision P se réduit à un problème de décision P' s'il existe un algorithme polynomial qui transforme toute entrée u de P en une entrée $u' = R(u)$ de P' telle que: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.



Si on connait 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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.



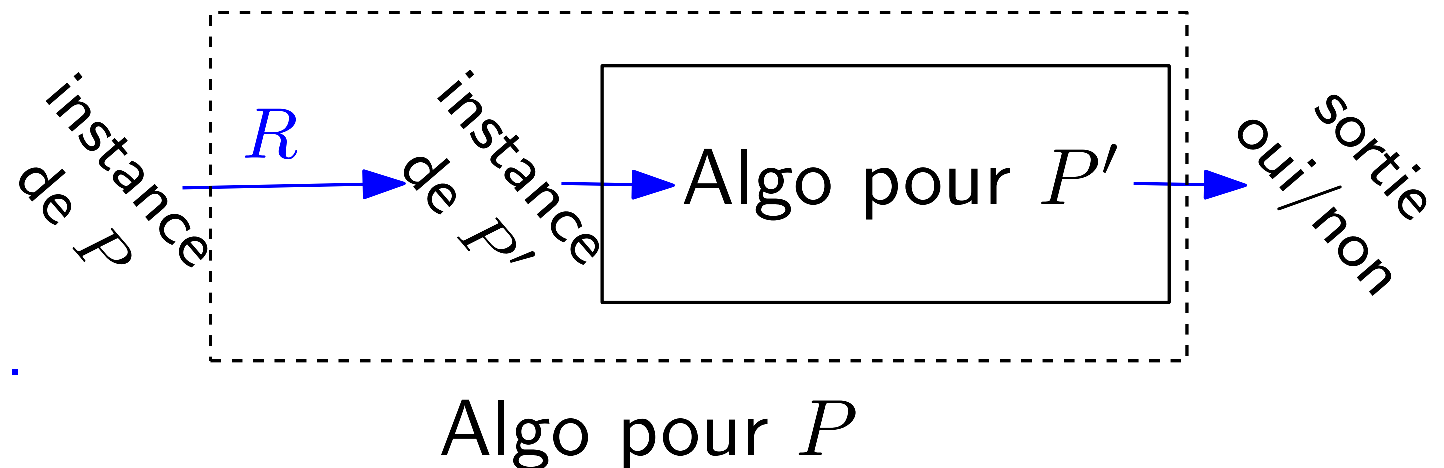
Si on connait vraiment un algo polynomial pour P', alors $P \in \mathcal{P}$.

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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.

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

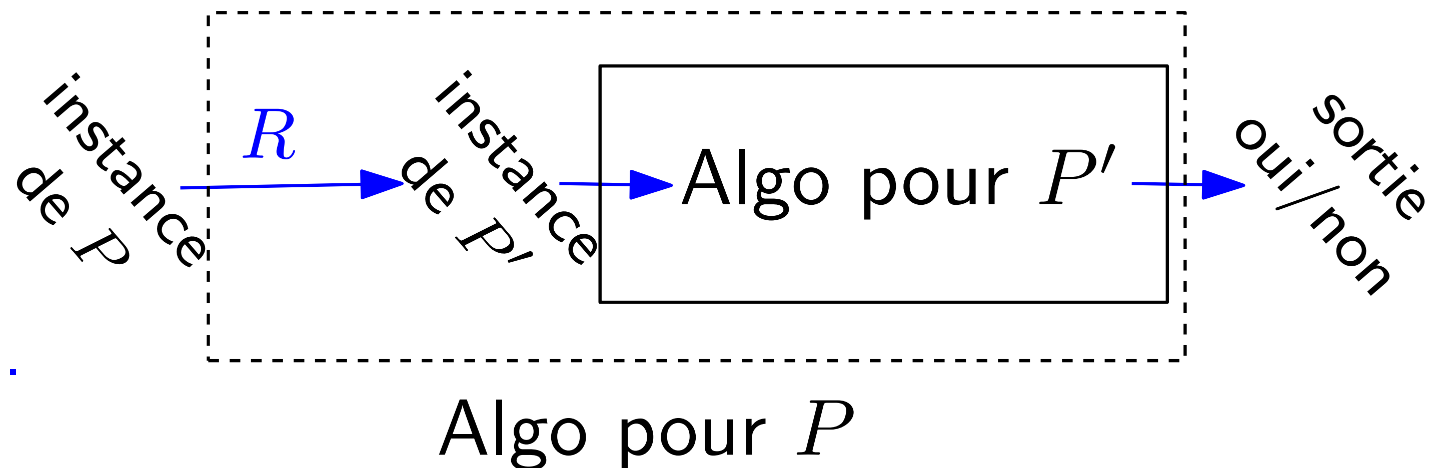


Si on connait vraiment un algo polynomial pour P', alors $P \in \mathcal{P}$.
Si on savait que P est dur ($P \notin \mathcal{P}$) alors P' aussi ($P' \notin \mathcal{P}$).

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: $u \in \mathrm{Oui}(P) \Leftrightarrow u' \in \mathrm{Oui}(P')$.

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



Si on connait 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...

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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

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:

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

Résultat admis: la preuve consiste en un codage de la 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:

$\Rightarrow$ 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:

$\Rightarrow$ 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...

Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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

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

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

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

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

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

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

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-SAT$\in \mathcal{P}$)

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

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

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 $C = x \vee y \vee z \vee t \vee u \vee \ldots$... par une conjonction de clause à 3 littéraux *connectées* par de nouvelles variables:

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

3-SAT est $\mathcal{NP}$-complet. (Remarque: 2-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 $C = x \vee y \vee z \vee t \vee u \vee \dots\dots$ 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-SAT$\in \mathcal{P}$)

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

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

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 $C = x \vee y \vee z \vee t \vee u \vee \ldots$... par une conjonction de clause à 3 littéraux *connectées* par de nouvelles variables:

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

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

CQFD

Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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 occurence 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 occurence 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 occurence 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' 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' a une clique de taille au moins k ssi G a un stable de taille au moins k.

Le problème COUVRANT:

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' a une clique de taille au moins k ssi G a un stable de taille au moins k.

Le problème COUVRANT:

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 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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 $V = \{x_1, \ldots, x_n\}$ et $E = \{e_0, \ldots, e_{m-1}\}$. 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 $V = \{x_1, \ldots, x_n\}$ et $E = \{e_0, \ldots, e_{m-1}\}$. 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 $e_{i_1}, \ldots, e_{i_p}$ on associe le nombre

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

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

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 $V = \{x_1, \ldots, x_n\}$ et $E = \{e_0, \ldots, e_{m-1}\}$. 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 $e_{i_1}, \ldots, e_{i_p}$ on associe le nombre

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

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

Observation. Pour tout choix de $X \subset V$ et $Y \subset \{0, \ldots, m-1\}$, la somme associée est de la forme $\sum_{j=0}^{m-1} u_i 4^i + v4^M$ avec $u_i \in \{0, 1, 2, 3\}$ et $v = |X|$.

Optimisation: le sac à dos

Donnée: Les poids $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 $p_1, \ldots, p_n$, les gains $a_1, \ldots, a_n$ et la capacité P.
Problème: Trouver $I \subset \{1, \ldots, n\}$ 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 (programmation dynamique): on parle de problème NP-complet au sens faible, par opposition aux problèmes NP-complet s au sens fort, qui restent NP-complet si les données numériques sont codées en unaire.

Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- 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
- Exemples de réductions plus évoluées

Alignement de k séquences

Donnée: k mots $u_1, \ldots, u_k$ sur un alphabet $\mathcal{A}$ et une fonction de dissimilarité des lettres $\delta : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}^+$.

Problème Alignement-Multiple:
trouver l'alignement $(U_1, \ldots, U_k)$ qui minimise

$$\sum_{1 \leq i < j \leq k} \delta(U_i, U_j).$$

Alignement de k séquences

Donnée: k mots $u_1, \ldots, u_k$ sur un alphabet $\mathcal{A}$ et une fonction de dissimilarité des lettres $\delta : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}^+$.

Problème Alignement-Multiple:
trouver l'alignement $(U_1, \ldots, U_k)$ qui minimise

$$\sum_{1 \leq i < j \leq k} \delta(U_i, U_j).$$

Programmation dynamique: par programmation dynamique on doit aligner des préfixes de chacuns des k mots

$\Rightarrow$ table en $O(n^k)$... exponentiel avec k.

Alignement de k séquences: NP-complétude

Théorème: Le problème de décision de l'existence d'un alignement multiple de valeur au plus S pour des séquences $u_1, \ldots, u_k$ et une fonction de dissimilarité δ est NP-complet.

Alignement de k séquences: NP-complétude

Théorème: Le problème de décision de l'existence d'un alignement multiple de valeur au plus S pour des séquences $u_1, \ldots, u_k$ et une fonction de dissimilarité δ est NP-complet.

Le problème ALIGNEMENT-MULTIPLE est clairement dans NP: le calcul de la valeur d'un alignement se fait en temps $O(k^2 n)$.

Alignement de k séquences: NP-complétude

Théorème: Le problème de décision de l'existence d'un alignement multiple de valeur au plus S pour des séquences $u_1, \ldots, u_k$ et une fonction de dissimilarité δ est NP-complet.

Le problème ALIGNEMENT-MULTIPLE est clairement dans NP: le calcul de la valeur d'un alignement se fait en temps $O(k^2 n)$.

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Alignement de k séquences: NP-complétude

Preuve: réduction de Vertex-cover à Alignement multiple.

Considérons une instance de VertexCover: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Considérons une instance de VERTEXCOVER: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de ALIGNEMENTMULTIPLE:

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Considérons une instance de VERTEXCOVER: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de ALIGNEMENTMULTIPLE:

- à l'arête $\{x_i, x_j\}$, $i < j$, on associe le mot de longueur $3n + 3$
$$u_{ij} = a^{3i-1} b a^{3(j-i)} b a^{3(n-j)+2}$$
avec des b en positions $3i$ et $3j + 1$.

Alignement de k séquences: NP-complétude

Preuve: réduction de Vertex-cover à Alignement multiple.

Considérons une instance de VertexCover: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de AlignementMultiple:

- à l'arête $\{x_i, x_j\}$, $i < j$, on associe le mot de longueur $3n + 3$
$$u_{ij} = a^{3i-1} b a^{3(j-i)} b a^{3(n-j)+2}$$
avec des b en positions $3i$ et $3j + 1$.
- on ajoute p copies du mot $t = c(001)^n 00c$ de longueur $3n + 4$ (1 en positions $3\ell{+}1$) et q copies de $v = cd^k c$ de longueur $k + 2$.

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Considérons une instance de VERTEXCOVER: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de ALIGNEMENTMULTIPLE:

- à l'arête $\{x_i, x_j\}$, $i < j$, on associe le mot de longueur $3n + 3$
$$u_{ij} = a^{3i-1} b a^{3(j-i)} b a^{3(n-j)+2}$$
avec des b en positions $3i$ et $3j + 1$.
- on ajoute p copies du mot $t = c(001)^n 00c$ de longueur $3n + 4$ (1 en positions $3\ell+1$) et q copies de $v = cd^k c$ de longueur $k + 2$.

Coûts: • si $x \neq y$, dans $\{a, b, 0, 1\}$, $\delta(x, y) = 1$ sauf $\delta(0, 1) = 2$

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Considérons une instance de VERTEXCOVER: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de ALIGNEMENTMULTIPLE:

- à l'arête $\{x_i, x_j\}$, $i < j$, on associe le mot de longueur $3n + 3$
$$u_{ij} = a^{3i-1}ba^{3(j-i)}ba^{3(n-j)+2}$$
avec des b en positions $3i$ et $3j + 1$.
- on ajoute p copies du mot $t = c(001)^n 00c$ de longueur $3n + 4$ (1 en positions $3\ell+1$) et q copies de $v = cd^k c$ de longueur $k + 2$.

Coûts: • si $x \neq y$, dans $\{a, b, 0, 1\}$, $\delta(x, y) = 1$ sauf $\delta(0, 1) = 2$

• si $y \in \{c, d\}$, $\delta(x, y) = 2$ sauf $\delta(a, c) = \delta(-, c) = \delta(1, d) = \delta(b, d) = 1$.

Alignement de k séquences: NP-complétude

Preuve: réduction de VERTEX-COVER à ALIGNEMENT MULTIPLE.

Considérons une instance de VERTEXCOVER: G un graphe à n sommets, m arêtes, et k l'objectif. Existe-t-il un ensemble d'au plus k sommets qui couvrent toutes les arêtes ?

On y associe une instance de ALIGNEMENTMULTIPLE:

- à l'arête $\{x_i, x_j\}$, $i < j$, on associe le mot de longueur $3n + 3$
$$u_{ij} = a^{3i-1}ba^{3(j-i)}ba^{3(n-j)+2}$$
avec des b en positions $3i$ et $3j + 1$.
- on ajoute p copies du mot $t = c(001)^n00c$ de longueur $3n + 4$ (1 en positions $3\ell+1$) et q copies de $v = cd^kc$ de longueur $k + 2$.

Coûts: • si $x \neq y$, dans $\{a, b, 0, 1\}$, $\delta(x, y) = 1$ sauf $\delta(0, 1) = 2$

• si $y \in \{c, d\}$, $\delta(x, y) = 2$ sauf $\delta(a, c) = \delta(-, c) = \delta(1, d) = \delta(b, d) = 1$.

On peut aligner un b de chaque u_{ij} avec des d ssi G possède une couverture avec k sommets, et ce cas donne l'alignement min.

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.
3-SAT se réduit à 3-NAESAT: soit $(C_i = x_i \vee y_i \vee z_i)_{i=1,\ldots,m}$ une instance de 3-SAT; on pose $C_i' = x_i \vee y_i \vee t_i$ et $C_i'' = z_i \vee \bar{t}_i \vee u$ où u et les t_i sont de nouvelles variables.

NP-complétude de $\textsc{MaxCut}$

On réduit 3-SAT à $\textsc{MaxCut}$ via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-SAT se réduit à 3-NAESAT: soit $(C_i = x_i \vee y_i \vee z_i)_{i=1,\ldots,m}$ une instance de 3-SAT; on pose $C_i' = x_i \vee y_i \vee t_i$ et $C_i'' = z_i \vee \bar{t}_i \vee u$ où u et les t_i sont de nouvelles variables.

Il faut voir que (C_i) admet une affectation ssi (C_i', C_i'') admet une bi-affectation (*ie* chaque clause a un littéral $\textsc{Vrai}$ et un $\textsc{Faux}$):

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-SAT se réduit à 3-NAESAT: soit $(C_i = x_i \vee y_i \vee z_i)_{i=1,\ldots,m}$ une instance de 3-SAT; on pose $C'_i = x_i \vee y_i \vee t_i$ et $C''_i = z_i \vee \bar{t}_i \vee u$ où u et les t_i sont de nouvelles variables.

Il faut voir que (C_i) admet une affectation ssi (C'_i, C''_i) admet une bi-affectation (*ie* chaque clause a un littéral Vrai et un Faux):

• s'il existe une affectation telle que tous les C_i soient vraies, on la complète par $t_i = \overline{x_i \vee y_i}$ et $u =$ Faux: c'est une bi-affectation.

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-SAT se réduit à 3-NAESAT: soit $(C_i = x_i \vee y_i \vee z_i)_{i=1,\dots,m}$ une instance de 3-SAT; on pose $C_i' = x_i \vee y_i \vee t_i$ et $C_i'' = z_i \vee \bar{t}_i \vee u$ où u et les t_i sont de nouvelles variables.

Il faut voir que (C_i) admet une affectation ssi (C_i', C_i'') admet une bi-affectation (*ie* chaque clause a un littéral Vrai et un Faux):

- s'il existe une affectation telle que tous les C_i soient vraies, on la complète par $t_i = \overline{x_i \vee y_i}$ et $u =$ Faux: c'est une bi-affectation.
- s'il existe une bi-affectation de (C_i', C_i''):
  - soit $u =$ Faux, alors $z_i \vee \bar{t}$ est vraie donc $x_i \vee y_i \vee z_i$ est vraie.
  - soit $u =$ Vrai, alors nier toutes les valeurs des variables...

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à MaxCut: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m + n$ arêtes).

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à MaxCut: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m + n$ arêtes).
Il faut voir que (C_i) admet une bi-affectation ssi le graphe associé admet une coupe avec $2m + n$ arêtes.

NP-complétude de $\textsc{MaxCut}$

On réduit 3-SAT à $\textsc{MaxCut}$ via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à $\textsc{MaxCut}$: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m+n$ arêtes).
Il faut voir que (C_i) admet une bi-affectation ssi le graphe associé admet une coupe avec $2m+n$ arêtes.
• Si on a une bi-affectation, on prend X_+ les sommets associés aux littéraux $\textsc{Vrai}$: dans la coupe, 2 arêtes par triangle et les $(x,\bar{x})$.

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à MaxCut: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m + n$ arêtes).
Il faut voir que (C_i) admet une bi-affectation ssi le graphe associé admet une coupe avec $2m + n$ arêtes.
• Si on a une bi-affectation, on prend X_+ les sommets associés aux littéraux Vrai: dans la coupe, 2 arêtes par triangle et les $(x, \bar{x})$.
• Une coupe de taille au moins $2m + n$ ne peut contenir plus de 2 arêtes de chaque triangle, donc elle en contient exactement 2.

NP-complétude de MAXCUT

On réduit 3-SAT à MAXCUT via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à MAXCUT: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m+n$ arêtes).
Il faut voir que (C_i) admet une bi-affectation ssi le graphe associé admet une coupe avec $2m+n$ arêtes.

- Si on a une bi-affectation, on prend X_+ les sommets associés aux littéraux VRAI: dans la coupe, 2 arêtes par triangle et les $(x, \bar{x})$.
- Une coupe de taille au moins $2m+n$ ne peut contenir plus de 2 arêtes de chaque triangle, donc elle en contient exactement 2. CQFD

NP-complétude de MaxCut

On réduit 3-SAT à MaxCut via le problème 3-NAESAT: étant donné un ensemble de clauses à 3 littéraux, trouver une affectation où chaque clause contient au moins un littéral vrai et un littéral faux.

3-NAESAT se réduit à MaxCut: à m clauses C_i sur n variables, on associe le graphe à $2n$ sommets (un par littéral), avec un triangle pour chaque clause et les arêtes x, $\bar{x}$ (soit $3m + n$ arêtes).
Il faut voir que (C_i) admet une bi-affectation ssi le graphe associé admet une coupe avec $2m + n$ arêtes.
• Si on a une bi-affectation, on prend X_+ les sommets associés aux littéraux Vrai: dans la coupe, 2 arêtes par triangle et les $(x, \bar{x})$.
• Une coupe de taille au moins $2m + n$ ne peut contenir plus de 2 arêtes de chaque triangle, donc elle en contient exactement 2. CQFD
Pour que cette réduction marche bien il faut d'abord se réduire à une variante de 3-NAESAT où 2 clauses partagent au plus une variable.

Cours 5: Réduction et NP-complétude

- Réduction entre problèmes
- Classes P, NP et NP-complétude
- Exemples de problèmes NP-complets.

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