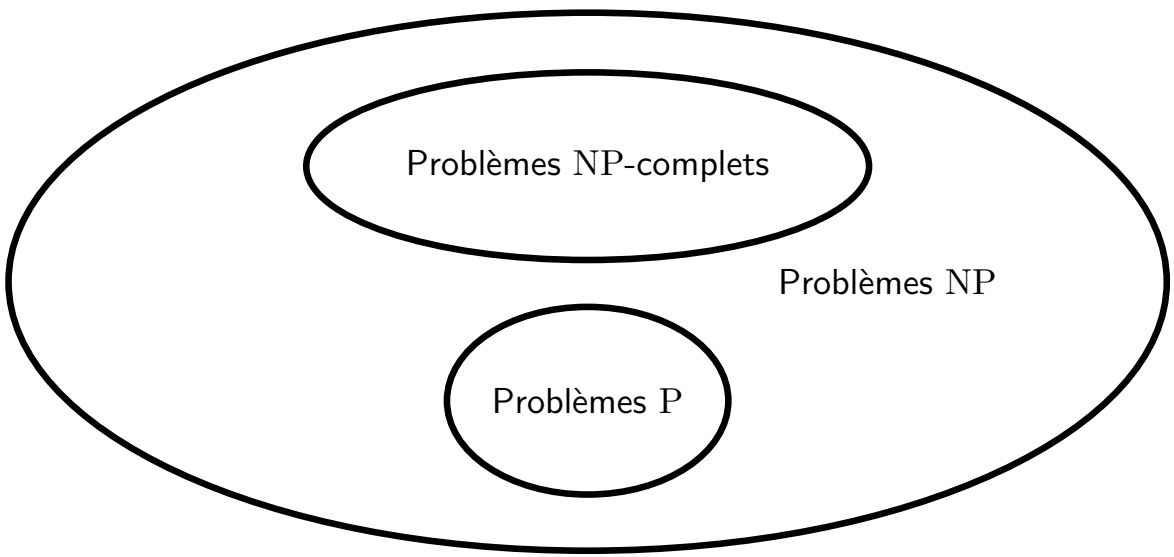
Algorithmes d'approximation

Plan

Les classes NP et P

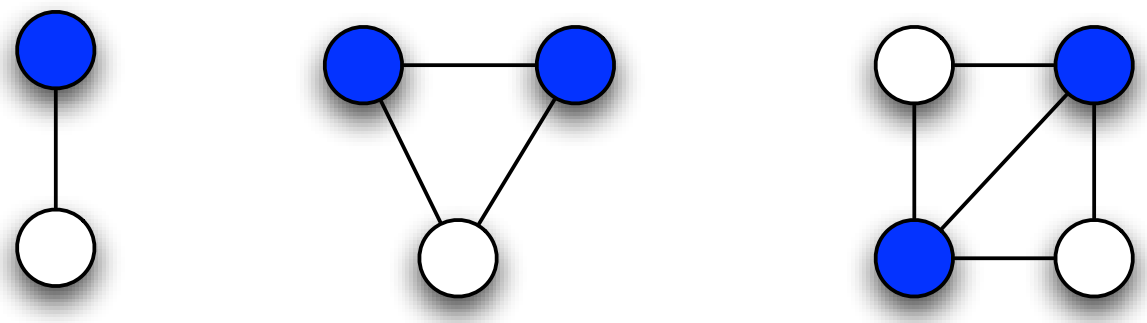
- Un problème de décision $\Pi = (D_\Pi, OUI_\Pi)$ correspond à un ensemble d'instances D_Π et à un sous-ensemble $OUI_\Pi \subset D_\Pi$ d'instances positives.

- Nous avons défini des classes NP et P :



Couverture de sommets

Le problème Couverture de sommets est NP-complet.
Données : un graphe non-orienté $G = (V, E)$ et un entier k.
Question : G contient-il une *couverture de sommets* $\mathcal{S}$ de cardinalité au plus k ?



Les sommets en bleu font partie de la couverture de sommets

Plan

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.
- De nombreux problèmes d'optimisation sont NP-complets.

Pourtant, il faut savoir les résoudre ! ! !

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.
- De nombreux problèmes d'optimisation sont NP-complets.

  Pourtant, il faut savoir les résoudre ! ! !
- Nous ne savons pas concevoir un algorithme polynomial si le problème est NP-complet.

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.
- De nombreux problèmes d'optimisation sont NP-complets.

  Pourtant, il faut savoir les résoudre ! ! !
- Nous ne savons pas concevoir un algorithme polynomial si le problème est NP-complet.
- Trois approches de résolutions peuvent contourner la difficulté :

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.
- De nombreux problèmes d'optimisation sont NP-complets.

  Pourtant, il faut savoir les résoudre ! ! !
- Nous ne savons pas concevoir un algorithme polynomial si le problème est NP-complet.
- Trois approches de résolutions peuvent contourner la difficulté :
  1. un algo. exponentiel si la taille de l'instance est petite.

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.
- De nombreux problèmes d'optimisation sont NP-complets.

  Pourtant, il faut savoir les résoudre ! ! !
- Nous ne savons pas concevoir un algorithme polynomial si le problème est NP-complet.
- Trois approches de résolutions peuvent contourner la difficulté :
  1. un algo. exponentiel si la taille de l'instance est petite.
  2. un algorithme optimal pour des instances particulières (importantes)

Comment résoudre un pb. optimisation ?

- Problème d'optimisation : trouver parmi les solutions d'un problème celle qui optimise une fonction f.

- De nombreux problèmes d'optimisation sont NP-complets.

  Pourtant, il faut savoir les résoudre ! ! !

- Nous ne savons pas concevoir un algorithme polynomial si le problème est NP-complet.

- Trois approches de résolutions peuvent contourner la difficulté :
  1. un algo. exponentiel si la taille de l'instance est petite.
  2. un algorithme optimal pour des instances particulières (importantes)
  3. un algo. polynomial trouvant une solution *"presque"* optimale.

Par exemple : le problème de couverture de sommets.

Entrée : un graphe $G = (V, E)$
Sortie : un ensemble de sommets
Algorithme :

Par exemple : le problème de couverture de sommets.

Entrée : un graphe $G = (V, E)$
Sortie : un ensemble de sommets
Algorithme :

1. Calculer un couplage maximal M de G ;

Par exemple : le problème de couverture de sommets.

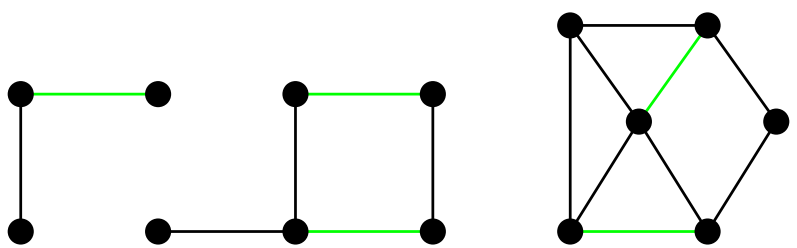
Entrée : un graphe $G = (V, E)$
Sortie : un ensemble de sommets
Algorithme :

1. Calculer un couplage maximal M de G ;

Un couplage M est un ensemble d'arêtes 2 à 2 non adjacentes :

$$\forall (a, a') \in M^2, \qquad a \neq a' \Rightarrow a \cap a' = \varnothing \ .$$

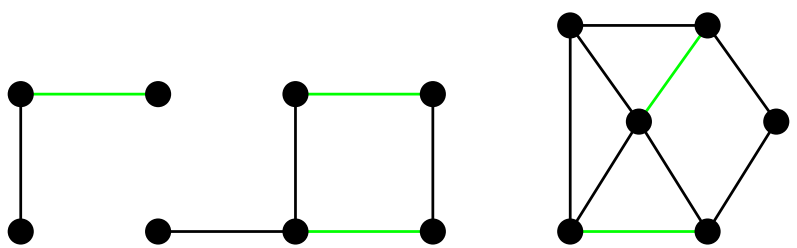
Un couplage maximal est un couplage M ayant la propriété que si une arête e est ajoutée, alors $M \cup \{e\}$ n'en est pas.



Par exemple : le problème de couverture de sommets.

Entrée : un graphe $G = (V, E)$
Sortie : un ensemble de sommets
Algorithme :

1. Calculer un couplage maximal M de G ;
2. Renvoyer l'ensemble des extrémités des arêtes du couplage.



Evaluation de la qualité de la solution

Soit OPT la cardinalité de la plus petite couverture de sommets dans G.

Evaluation de la qualité de la solution

Soit OPT la cardinalité de la plus petite couverture de sommets dans G.

Théorème (qualité de la solution)

L'algorithme retourne une couverture de sommets S telle que

$$|S| \leq 2 \cdot OPT.$$

Evaluation de la qualité de la solution

Soit OPT la cardinalité de la plus petite couverture de sommets dans G.

Théorème (qualité de la solution)

L'algorithme retourne une couverture de sommets S telle que

$$|S| \leq 2 \cdot OPT.$$

Preuve :

- Toutes les arêtes sont couvertes par l'ensemble S de sommets renvoyés
- Si ce n'est pas le cas, l'arête non couverte pourrait être ajoutée au couplage
  $\Longrightarrow$ contradiction avec le fait que le couplage soit maximal.

Evaluation de la qualité de la solution

Soit OPT la cardinalité de la plus petite couverture de sommets dans G.

Théorème (qualité de la solution)

L'algorithme retourne une couverture de sommets S telle que

$$|S| \leq 2 \cdot OPT.$$

Preuve :

- Toutes les arêtes sont couvertes par l'ensemble S de sommets renvoyés
- Soit M le couplage sélectionné.

  Toute couverture de sommets doit couvrir les arêtes de M

  $|M| \leq OPT$

□

Evaluation de la qualité de la solution

Soit OPT la cardinalité de la plus petite couverture de sommets dans G.

Théorème (qualité de la solution)

L'algorithme retourne une couverture de sommets S telle que

$$|S| \leq 2 \cdot OPT.$$

Preuve :

- Toutes les arêtes sont couvertes par l'ensemble S de sommets renvoyés
- Soit M le couplage sélectionné.

  Toute couverture de sommets doit couvrir les arêtes de M
  $|M| \leq OPT$
- La taille de la couverture renvoyée est $2|M| (\leq 2OPT)$.

□

Définitions.

Un problème d'optimisation combinatoire de minimisation Π est défini de la facon suivante :

- D_Π : un ensemble d'instances.
- $S_\Pi(I)$: un ensemble fini non-vide de solutions pour chaque instance I de D_Π
- une fonction d'évaluation m_Π

Pour chaque instance I de D_Π, et
pour chaque solution $\sigma \in S_\Pi(I)$:
$m_\Pi(I, \sigma)$ est la valeur de la solution (un rationnel).

Définitions.

Un problème d'optimisation combinatoire de minimisation Π est défini de la facon suivante :

- D_Π : un ensemble d'instances.

  D_{VC} = ens. des graphes

- $S_\Pi(I)$: un ensemble fini non-vide de solutions pour chaque instance I de D_Π

  $S_{VC}(I)$ = ens. des couvertures sommets

- une fonction d'évaluation m_Π

  $m_{VC}(I, \sigma)$ = cardinalité de la solution σ

Pour chaque instance I de D_Π, et
pour chaque solution $\sigma \in S_\Pi(I)$:
$m_\Pi(I, \sigma)$ est la valeur de la solution (un rationnel).

Notations

Soit Π un problème d'optimisation. Soit I une instance de D_Π

- Un algorithme A est un algorithme d'optimisation de Π si pour toute instance I de D_Π, A retourne une solution $\sigma \in S_\Pi(I)$.
- Notons

  $A(I) = m_\Pi(I, \sigma)$ pour la solution σ retournée.

  $OPT_\Pi(I) = m_\Pi(I, \sigma^*)$ pour une solution optimale σ^*.

- Parfois, les notations OPT, $OPT(I)$ sont utilisées pour désigner la valeur de la solution optimale.

Le facteur d'approximation

Problème : Trouver parmi les solutions d'un problème Π celle qui optimise une fonction m_Π.
Un algorithme A approche l'optimum à un facteur α si

$$\text{pour toute instance } I, \text{ on a } max\left(\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right) \leq \alpha$$

- Pour un problème de minimisation :
$$OPT(I) \leq A(I) \leq \alpha OPT(I)$$
- Pour un problème de maximisation :
$$(1/\alpha)OPT(I) \leq A(I) \leq OPT(I)$$

Le facteur d'approximation

Problème : Trouver parmi les solutions d'un problème Π celle qui optimise une fonction m_Π.
Un algorithme A approche l'optimum à un facteur α si

$$\text{pour toute instance } I, \text{ on a } max\left(\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right) \leq \alpha$$

L'objectif est que α soit proche de 1

Problème Couverture de sommets : récapitulatif

Résultats :

- L'algorithme approche l'optimum à un facteur 2.

  C'est un algorithme polynomial ayant un facteur d'approximation 2.

Problème Couverture de sommets : récapitulatif

Résultats :

- L'algorithme approche l'optimum à un facteur 2.

  C'est un algorithme polynomial ayant un facteur d'approximation 2.

Définition :

- La classe des problème d'optimisation pour lesquels il existe un algorithme polynomial d'approximation à un facteur borné est notée APX.

Problème Couverture de sommets : récapitulatif

Résultats :

- L'algorithme approche l'optimum à un facteur 2.

  C'est un algorithme polynomial ayant un facteur d'approximation 2.
- Le problème Couverture de sommets est dans APX.

Définition :

- La classe des problème d'optimisation pour lesquels il existe un algorithme polynomial d'approximation à un facteur borné est notée APX.

Plan

Rappel : problème du stockage.

Considérons n programmes $P_1, P_2, \ldots, P_n$ qui peuvent être stockés sur un disque dur de capacité D gigabytes.

- Chaque programme P_i a besoin s_i gigabytes pour être stocké.
- Tous les programmes ne peuvent pas être stockés sur le disque : $(\sum_{i=1}^{n} s_i > D)$

Objectif :

Concevoir un algorithme qui permet de maximiser **la quantité de données stockées** sur le disque.

Algorithme dynamique

Entrée : $\begin{cases} \mathcal{P} : \text{ un ens. de programmes : } \mathcal{P} = \{P_1, P_2, \ldots, P_n\} \\ \text{tel que chaque programme } P_i \text{ est de taille } s_i \\ D : \text{un entier} \end{cases}$

Sortie : un entier

1. pour j allant de 1 à D faire $T_0[j] \leftarrow 0$
2. $T_0[0] \leftarrow 1$ // la solution $S = \emptyset$ est une solution faisable
3. pour i allant de 1 à $|\mathcal{P}|$ faire
   3.1 pour j allant de 1 à D faire
      3.1.1 si $T_{i-1}[j] == 1$ alors $\begin{cases} T_i[j] \leftarrow 1 \\ T_i[j + s_i] \leftarrow 1 : \text{ si } j + s_i \leq D \end{cases}$
4. retourner la plus grande valeur j telle que $T_n[j] == 1$

Complexité : $\mathcal{O}(D \cdot |\mathcal{P}|)$

Retour au cours précédent

- Le problème du stockage peut se résoudre en temps polynomial si les entiers de l'instance sont codés en unaire.
- En fait :

Théorème

Le problème du STOCKAGE est NP-complet si les entiers sont codés en base 2.

Retour au cours précédent

- Le problème du stockage peut se résoudre en temps polynomial si les entiers de l'instance sont codés en unaire.
- En fait :

Théorème

Le problème du STOCKAGE est NP-complet si les entiers sont codés en base 2.

◂ Début de la preuve

Retour au cours précédent

- Le problème du stockage peut se résoudre en temps polynomial si les entiers de l'instance sont codés en unaire.
- En fait :

Théorème

Le problème du STOCKAGE est NP-complet si les entiers sont codés en base 2.

Définitions :

- Un problème NP-complet est **faiblement NP-complet** si il est NP-complet avec les entiers de l'instance codés en binaire.

Retour au cours précédent

- Le problème du stockage peut se résoudre en temps polynomial si les entiers de l'instance sont codés en unaire.
- En fait :

Théorème

Le problème du STOCKAGE est NP-complet si les entiers sont codés en base 2.

Définitions :

- Un problème NP-complet est **faiblement NP-complet** si il est NP-complet avec les entiers de l'instance codés en binaire.

- Un problème NP-complet est **fortement NP-complet** si il est NP-complet avec les entiers de l'instance codés en unaire.

Problème du STOCKAGE : Algorithme approximation

Idée 1 : Réduire la complexité (en nombre d'opérations) de l'algorithme dynamique.

stoquer toutes les solutions dans une liste.

Entrée : : un ensemble fini d'entiers $\mathcal{P} = \{s_1, s_2, \ldots, s_n\}$ et un entier $D \in \mathbb{N}$

Sortie : un entier

1. $L_0 \leftarrow < 0 >$
2. pour i à allant de 1 à $|\mathcal{P}|$
   - 2.1 $L_i \leftarrow fusion - liste(L_{i-1}, L_{i-1} \oplus s_i)$
   - 2.2 Supprimer tous les éléments de L_i qui sont supérieurs à D
3. retourner le plus grand élément de $L_{|\mathcal{P}|}$

Problème du STOCKAGE : Algorithme approximation

Idée 1 : Réduire la complexité (en nombre d'opérations) de l'algorithme dynamique.

stoquer toutes les solutions dans une liste.

Entrée : : un ensemble fini d'entiers $\mathcal{P} = \{s_1, s_2, \ldots, s_n\}$ et un entier $D \in \mathbb{N}$

Sortie : un entier

1. $L_0 \leftarrow < 0 >$
2. pour i à allant de 1 à $|\mathcal{P}|$
   - 2.1 $L_i \leftarrow fusion - liste(L_{i-1}, L_{i-1} \oplus s_i)$
   - 2.2 Supprimer tous les éléments de L_i qui sont supérieurs à D
3. retourner le plus grand élément de $L_{|\mathcal{P}|}$

Notation : $L \oplus x$ est la liste L où chaque élement est incrémenté par x.

Récapitulatif : L'algorithme retourne la solution optimale en $\mathcal{O}(D \cdot |\mathcal{P}|)$ opérations.

Problème du STOCKAGE : Algorithme approximation

Remarque : La complexité de l'algorithme dynamique dépend du nombre d'éléments dans $L_{|\mathcal{P}|}$.

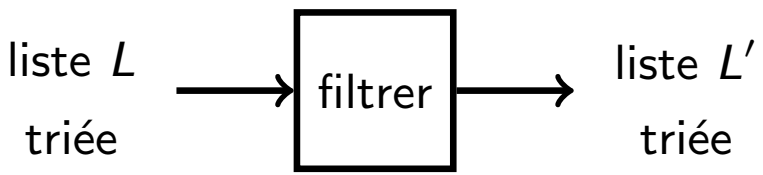
Idée 2 : Réduire le nombre d'éléments stockés dans les listes L_i sans trop dégrader la qualité de la solution.

Problème du STOCKAGE : Algorithme approximation

Remarque : La complexité de l'algorithme dynamique dépend du nombre d'éléments dans $L_{|\mathcal{P}|}$.

Idée 2 : Réduire le nombre d'éléments stockés dans les listes L_i sans trop dégrader la qualité de la solution.

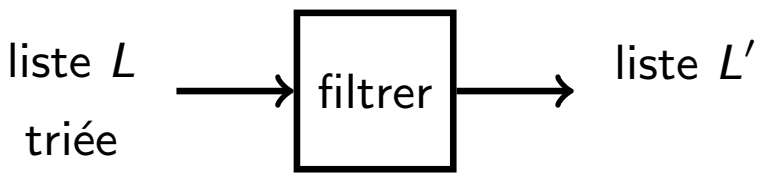
Filtrer les listes



Tous les éléments p de L ne sont pas dans L'
s'il existe un élément z dans L' qui *approxime* p, c'est-à-dire :
$\frac{p}{1+\delta} \leq z \leq p$

Exemple sur le filtrage

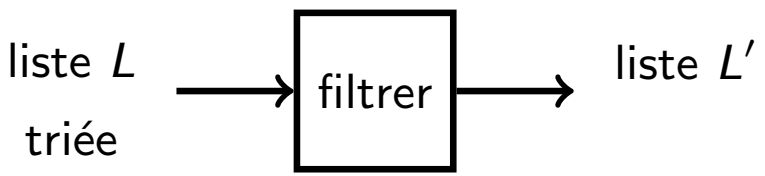
Filtrer les listes avec $\delta = 0,1$ comme seuil



- En entrée : $L =< 10; 11; 12; 15; 20; 21; 22; 23; 24; 29 >$

Exemple sur le filtrage

Filtrer les listes avec $\delta = 0,1$ comme seuil

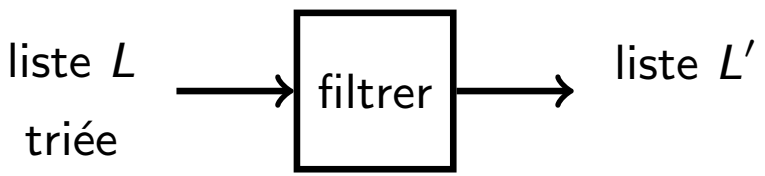


- En entrée : $L =< 10; 11; 12; 15; 20; 21; 22; 23; 24; 29 >$

l'élément 11 peut être supprimé car
10 *approxime* 11 dans L'
c'est-à-dire : $\frac{11}{1+\delta} \leq 10 \leq 11$

Exemple sur le filtrage

Filtrer les listes avec $\delta = 0,1$ comme seuil



- En entrée : $L =< 10; 11; 12; 15; 20; 21; 22; 23; 24; 29 >$

  l'élément 11 peut être supprimé car
  10 *approxime* 11 dans L'
  c'est-à-dire : $\frac{11}{1+\delta} \leq 10 \leq 11$

- En sortie : $L' =< 10; 12; 15; 20; 23; 29 >$

Algorithme en utilisant le filtrage

Entrée : un ensemble fini d'entiers $\mathcal{P} = \{s_1, s_2, \ldots, s_n\}$, un entier $D \in \mathbb{N}$, un entier $0 < \delta \leq 1$
Sortie : un entier

1. $L_0 \leftarrow < 0 >$
2. pour i à allant de 1 à $|\mathcal{P}|$
   2.1 $L_i \leftarrow fusion - liste(L_{i-1}, L_{i-1} \oplus s_i)$;
   2.3 Supprimer tous les éléments de L_i qui sont supérieurs à D ;
3. retourner le plus grand élément de $L_{|\mathcal{P}|}$

Algorithme en utilisant le filtrage

Entrée : un ensemble fini d'entiers $\mathcal{P} = \{s_1, s_2, \ldots, s_n\}$, un entier $D \in \mathbb{N}$, un entier $0 < \delta \leq 1$
Sortie : un entier

1. $L_0 \leftarrow < 0 >$
2. pour i à allant de 1 à $|\mathcal{P}|$
   - 2.1 $L_i \leftarrow fusion - liste(L_{i-1}, L_{i-1} \oplus s_i)$;
   - 2.2 filtrer la liste L_i utilisant la valeur δ ;
   - 2.3 Supprimer tous les éléments de L_i qui sont supérieurs à D ;
3. retourner le plus grand élément de $L_{|\mathcal{P}|}$

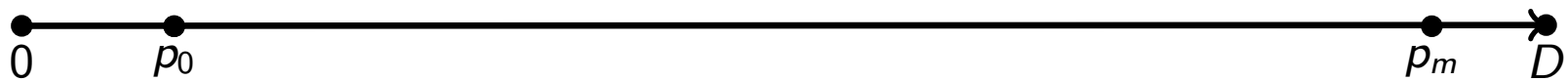
Complexité de l'algorithme

Lemme

L'algorithme est en donc polynomial en la taille des données et en $1/\delta$.

Preuve : On majore le nombre d'éléments de $L_{|\mathcal{P}|} = \{p_0, \ldots, p_m\}$.

1. $D \geq p_m \geq (1+\delta)^{m-1} \cdot p_0 \geq (1+\delta)^{m-1}$



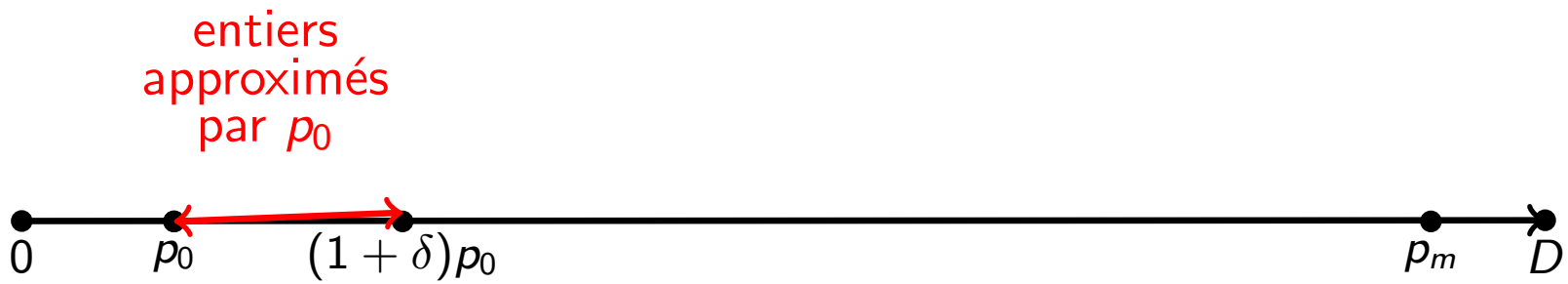
Complexité de l'algorithme

Lemme

L'algorithme est en donc polynomial en la taille des données et en $1/\delta$.

Preuve : On majore le nombre d'éléments de $L_{|\mathcal{P}|} = \{p_0, \ldots, p_m\}$.

1. $D \geq p_m \geq (1+\delta)^{m-1} \cdot p_0 \geq (1+\delta)^{m-1}$



□

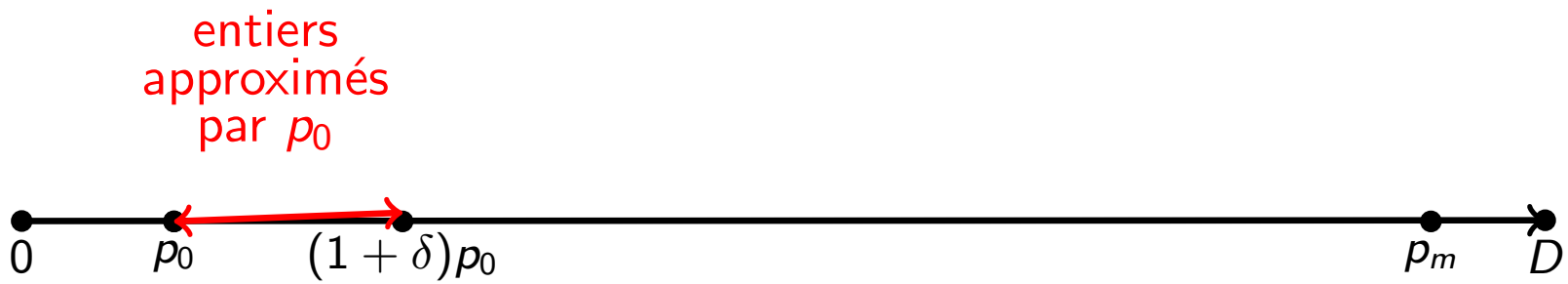
Complexité de l'algorithme

Lemme

L'algorithme est en donc polynomial en la taille des données et en $1/\delta$.

Preuve : On majore le nombre d'éléments de $L_{|\mathcal{P}|} = \{p_0, \ldots, p_m\}$.

1. $D \geq p_m \geq (1+\delta)^{m-1} \cdot p_0 \geq (1+\delta)^{m-1}$



2. d'où $\frac{\ln D}{\ln(1+\delta)} \geq m-1$
3. $\ln D \cdot \frac{1+\delta}{\delta} + 1 \geq m$ comme $\ln(1+\delta) \geq \frac{\delta}{1+\delta}$
4. m est donc polynomial en $\ln D$ et en $1/\delta$

□

facteur d'approximation

Lemme

Soit $\varepsilon \geq 0$. En posant $\delta = \frac{\varepsilon}{2n}$. Le facteur d'approximation de l'algorithme est à un facteur $1 + \varepsilon$.

Preuve : On veut prouver que :

$$\frac{OPT}{(1+\varepsilon)} \leq maxL_{|\mathcal{P}|} \leq OPT$$

- Soit S_i l'ensemble des sommes partielles de $\mathcal{P}_i = \{s_1, \ldots, s_i\}$ inférieures à D.

facteur d'approximation

Lemme

Soit $\varepsilon \geq 0$. En posant $\delta = \frac{\varepsilon}{2n}$. Le facteur d'approximation de l'algorithme est à un facteur $1 + \varepsilon$.

Preuve : On veut prouver que :

$$\frac{OPT}{(1+\varepsilon)} \leq maxL_{|\mathcal{P}|} \leq OPT$$

- Soit S_i l'ensemble des sommes partielles de $\mathcal{P}_i = \{s_1, \ldots, s_i\}$ inférieures à D.
- On peut prouver par récurrence sur i

$$\forall p \in S_i, \exists y \in L_i, \frac{p}{(1+\delta)^i} \leq y \leq p$$

facteur d'approximation

Lemme

Soit $\varepsilon \geq 0$. En posant $\delta = \frac{\varepsilon}{2n}$. Le facteur d'approximation de l'algorithme est à un facteur $1 + \varepsilon$.

Preuve : On veut prouver que :

$$\frac{OPT}{(1+\varepsilon)} \leq maxL_{|\mathcal{P}|} \leq OPT$$

- Soit S_i l'ensemble des sommes partielles de $\mathcal{P}_i = \{s_1, \ldots, s_i\}$ inférieures à D.
- On peut prouver par récurrence sur i

$$\forall p \in S_i, \exists y \in L_i, \frac{p}{(1+\delta)^i} \leq y \leq p$$

- Grâce à l'hypothèse de récurrence, on obtient :

$$\frac{OPT}{(1+\frac{\varepsilon}{2n})^n} \leq maxL_{|\mathcal{P}|}$$

en posant $p = OPT$ et $y = maxL_{|\mathcal{P}|}$

facteur d'approximation

Lemme

Soit $\varepsilon \geq 0$. En posant $\delta = \frac{\varepsilon}{2n}$. Le facteur d'approximation de l'algorithme est à un facteur $1 + \varepsilon$.

Preuve : On veut prouver que :

$$\frac{OPT}{(1+\varepsilon)} \leq maxL_{|\mathcal{P}|} \leq OPT$$

- Soit S_i l'ensemble des sommes partielles de $\mathcal{P}_i = \{s_1, \ldots, s_i\}$ inférieures à D.
- On peut prouver par récurrence sur i

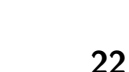
$$\forall p \in S_i, \exists y \in L_i, \frac{p}{(1+\delta)^i} \leq y \leq p$$

- Grâce à l'hypothèse de récurrence, on obtient :

$$\frac{OPT}{(1+\frac{\varepsilon}{2n})^n} \leq maxL_{|\mathcal{P}|}$$

en posant $p = OPT$ et $y = maxL_{|\mathcal{P}|}$

$\frac{OPT}{(1+\varepsilon)} \leq maxL_{|\mathcal{P}|}$ il suffit de prouver que $(1+\frac{\varepsilon}{2n})^n \leq (1+\varepsilon)$



Problème du STOCKAGE : récapitulatif

Résultats :

Problème du STOCKAGE : récapitulatif

Résultats :

- Un algorithme approche l'optimum à un facteur $1 + \varepsilon$.

Problème du STOCKAGE : récapitulatif

Résultats :

- Un algorithme approche l'optimum à un facteur $1 + \varepsilon$.

Définitions :

- Un problème admet un schéma d'approximation si pour tout ε fixé, il admet un algorithme ayant facteur d'approximation $1 + \varepsilon$.

Problème du STOCKAGE : récapitulatif

Résultats :

- Un algorithme approche l'optimum à un facteur $1 + \varepsilon$.
- La complexité est polynomiale en la taille et en $1/\varepsilon$.

Définitions :

- Un problème admet un schéma d'approximation si pour tout ε fixé, il admet un algorithme ayant facteur d'approximation $1 + \varepsilon$.
- Le schéma d'approximation est totalement polynomial si la complexité est polynomiale en la taille de l'instance et en $1/\varepsilon$.

La classe de problème correspondante s'appelle FPTAS

Problème du STOCKAGE : récapitulatif

Résultats :

- Un algorithme approche l'optimum à un facteur $1 + \varepsilon$.
- La complexité est polynomiale en la taille et en $1/\varepsilon$.

Définitions :

- Un problème admet un schéma d'approximation si pour tout ε fixé, il admet un algorithme ayant facteur d'approximation $1 + \varepsilon$.
- Le schéma d'approximation est totalement polynomial si la complexité est polynomiale en la taille de l'instance et en $1/\varepsilon$.

La classe de problème correspondante s'appelle FPTAS

Problème du STOCKAGE : récapitulatif

Résultats :

- Un algorithme approche l'optimum à un facteur $1 + \varepsilon$.
- La complexité est polynomiale en la taille et en $1/\varepsilon$.

STOCKAGE est dans FPTAS.

Définitions :

- Un problème admet un schéma d'approximation si pour tout ε fixé, il admet un algorithme ayant facteur d'approximation $1 + \varepsilon$.
- Le schéma d'approximation est totalement polynomial si la complexité est polynomiale en la taille de l'instance et en $1/\varepsilon$.

La classe de problème correspondante s'appelle FPTAS

Plan

Problème du voyageur de commerce (TSP)

Entrée : Etant donnés un ensemble de villes et une fonction distance entre les villes.
Objectif : trouver un cycle de coût minimum passant par chaque ville exactement une fois.

Remarque :

- Il existe un algorithme polynomial qui a un facteur d'approximation de 2 pour le TSP ayant une fonction de poids respectant l'inégalité triangulaire.

- Par contre, pour une fonction de poids arbitraire, le problème est difficile à résoudre.

Résultat d'inapproximabilité.

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieur à k à moins que $P = NP$.

Résultat d'inapproximabilité.

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieur à k à moins que $P = NP$.

Preuve : Raisonnons par l'absurde.

1. Soit $\mathcal{A}$ un algo. poly. ayant un facteur d'approximation $\alpha \leq k$.
2. Construisons un algorithme $\mathcal{F}$ qui résoud le problème du cycle hamiltonien (CH) ayant comme entrée le graphe $G = (V, E)$:
   - 2.1 L'ensemble de villes $= V$
   - 2.2 Distance entre u et $v = \begin{cases} 1 & si \quad (u, v) \in E \\ nk + 1 & sinon \end{cases}$

Résultat d'inapproximabilité.

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieur à k à moins que $P = NP$.

Preuve : Raisonnons par l'absurde.

1. Soit $\mathcal{A}$ un algo. poly. ayant un facteur d'approximation $\alpha \leq k$.
2. Construisons un algorithme $\mathcal{F}$ qui résoud le problème du cycle hamiltonien (CH) ayant comme entrée le graphe $G = (V, E)$:
   - 2.1 L'ensemble de villes $= V$
   - 2.2 Distance entre u et $v = \begin{cases} 1 & si \quad (u,v) \in E \\ nk+1 & sinon \end{cases}$

Remarque : Si G admet un cycle hamiltonien,

- alors le coût optimal du TSP dans K est n ;
- Sinon le coût optimal du TSP est $> kn$.

Résultat d'inapproximabilité.

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieur à k à moins que $P = NP$.

Preuve : Raisonnons par l'absurde.

1. Soit $\mathcal{A}$ un algo. poly. ayant un facteur d'approximation $\alpha \leq k$.
2. Construisons un algorithme $\mathcal{F}$ qui résoud le problème du cycle hamiltonien (CH) ayant comme entrée le graphe $G = (V, E)$:
   - 2.1 L'ensemble de villes $= V$
   - 2.2 Distance entre u et $v = \begin{cases} 1 & si \quad (u, v) \in E \\ nk + 1 & sinon \end{cases}$
   - 2.3 Si $\mathcal{A}$ calcule une solution de coût $\leq k \cdot n$,
     - alors retourner oui ; G admet un cycle hamiltonien,
     - sinon retourner non ; G n'admet pas un cycle hamiltonien,

Résultat d'inapproximabilité.

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieur à k à moins que $P = NP$.

Preuve : Raisonnons par l'absurde.

1. Soit $\mathcal{A}$ un algo. poly. ayant un facteur d'approximation $\alpha \leq k$.
2. Construisons un algorithme $\mathcal{F}$ qui résoud le problème du cycle hamiltonien (CH) ayant comme entrée le graphe $G = (V, E)$:
   - 2.1 L'ensemble de villes $= V$
   - 2.2 Distance entre u et $v = \begin{cases} 1 & si \quad (u, v) \in E \\ nk + 1 & sinon \end{cases}$
   - 2.3 Si $\mathcal{A}$ calcule une solution de coût $\leq k \cdot n$,
     - alors retourner oui ; G admet un cycle hamiltonien,
     - sinon retourner non ; G n'admet pas un cycle hamiltonien,
3. $\mathcal{F}$ est un algo poly. qui résoud le problème CH.

$\Longrightarrow$ contradiction avec $P \neq NP$

Technique d'inapproximation

Théorème

Soit $k \geq 1$ un entier, calculable en temps polynomial. Il n'existe pas d'algorithme polynomial pour le problème TSP ayant un facteur d'approximation inférieure à k à moins que $P = NP$.

- Il existe un technique permettant de trouver des résultats d'impossibilité.

la réduction écartante

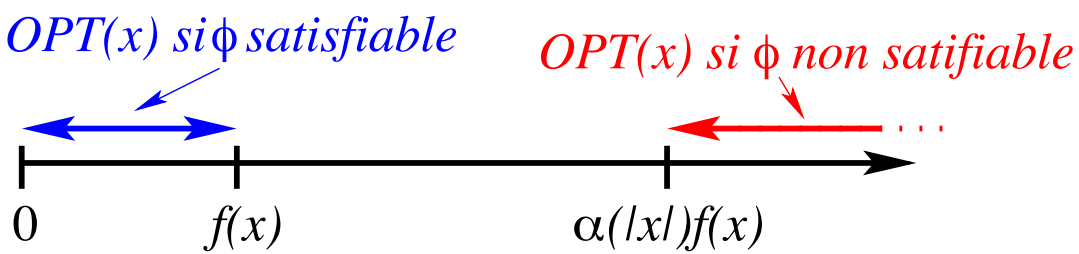
Définition d'une réduction.

Une réduction écartante $\mathcal{R}$ à partir de 3-SAT vers le problème de minimisation Π corresponds à 2 paramètres f et α.

La réduction transforme, en temps polynomial, une instance de 3-SAT, ϕ en une de Π noté x telle que

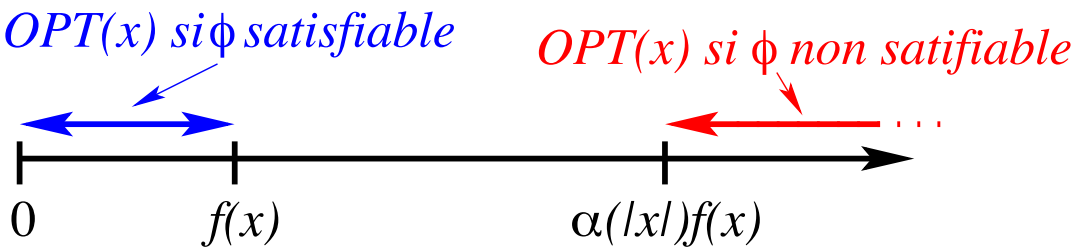
Si ϕ est satisfiable, $OPT(x) \leq f(x)$

Si ϕ n'est pas satisfiable, $OPT(x) > \alpha(|x|)f(x)$

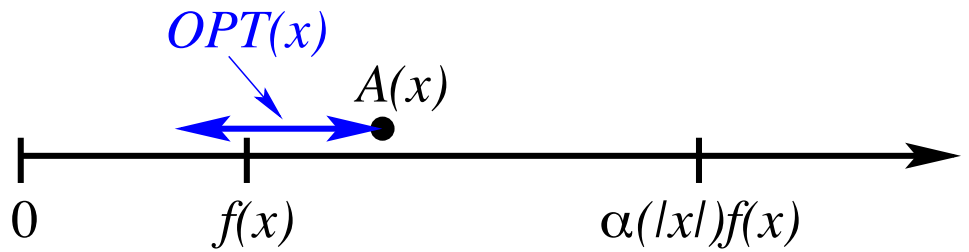


Illustration

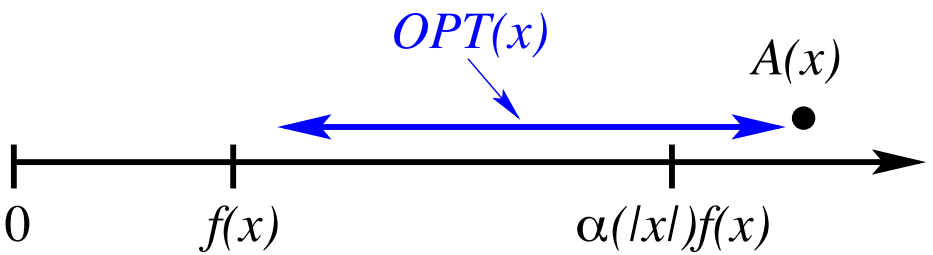
- Par réduction,



- Si $A(x) \leq \alpha(|x|)f(x)$, alors ϕ est satisfiable, car



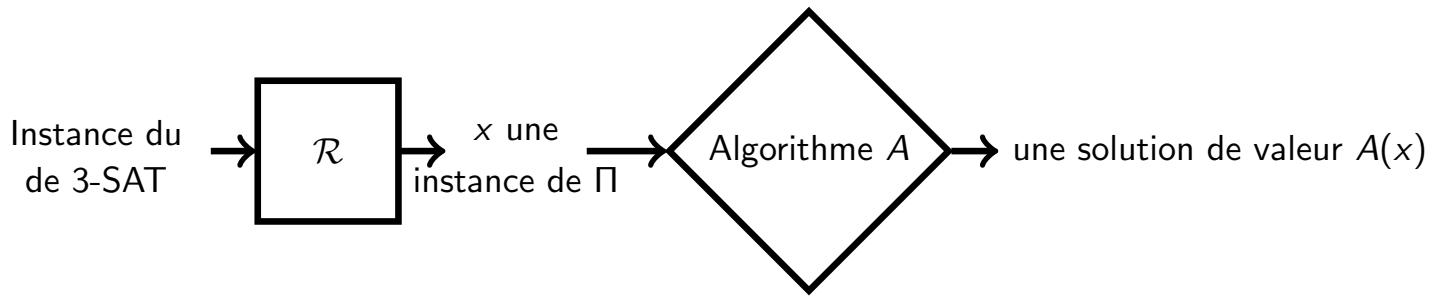
- Si $A(x) > \alpha(|x|)f(x)$, alors ϕ n'est pas satisfiable, car



Conséquence :

Remarque : La réduction $\mathcal{R}$ montre qu'il n'existe pas d'algorithme d'approximation de rapport $\alpha(|x|)$ si $P \neq NP$.

1. Soit A un algo. poly. d'approx. de rapport $r \leq \alpha(|x|)$ pour Π.
2. Construisons un algorithme $\mathcal{A}'$ pour 3-SAT



Si $A(x) \leq \alpha(|x|)f(x)$, retourner VRAI
sinon retourner FAUX .

3. L'algorithme $\mathcal{A}'$ résoud le problème 3-SAT en temps polynomial

$\Longrightarrow$ contradiction avec $P \neq NP$

Plan

Conclusion

- Notion récente d'algorithmes d'approximation (Graham 1966, Johnson 1974).
- Depuis, des milliers d'algorithmes d'approximation ont été trouvés pour un large panel de problèmes.
- Résultat d'impossibilité : limite de l'approximabilité.

ANNEXE

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

  Il faut parvenir à traduire deux contraintes : un sommet peut appartenir ou pas à la couverture et le sous-ensemble de sommets couvre bien toutes les arêtes.

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

- Pour cela,

  ◂ Exemple 1

  - on numérote les sommets entre 0 et $n-1$
    et des arêtes entre 0 et $m-1$.

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

- Pour cela,
  - on numérote les sommets entre 0 et $n-1$
    et des arêtes entre 0 et $m-1$.
  - on associe un entier b_{ij} pour chaque couple (arête,sommet)

$$b_{ij} = \begin{cases} 1 & \text{si l'arête } i \text{ est incidente au sommet } j, \\ 0 & \text{sinon} \end{cases}$$

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

- Pour cela,

◂ Exemple 2

  - on numérote les sommets entre 0 et $n-1$
    et des arêtes entre 0 et $m-1$.
  - on associe un entier b_{ij} pour chaque couple (arête,sommet)

$$b_{ij} = \begin{cases} 1 & \text{si l'arête } i \text{ est incidente au sommet } j, \\ 0 & \text{sinon} \end{cases}$$

- On construit l'ensemble $\mathcal{P}$ de la façon suivante : $(b = 4)$

  - Pour chaque sommet j : $s_j = b^m + \sum_{i=0}^{m-1} b_{ij} b^i$

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

- Pour cela,

◂ Exemple 3

  - on numérote les sommets entre 0 et $n-1$
    et des arêtes entre 0 et $m-1$.
  - on associe un entier b_{ij} pour chaque couple (arête,sommet)

$$b_{ij} = \begin{cases} 1 & \text{si l'arête } i \text{ est incidente au sommet } j, \\ 0 & \text{sinon} \end{cases}$$

- On construit l'ensemble $\mathcal{P}$ de la façon suivante : $(b = 4)$

  - Pour chaque sommet j : $s_j = b^m + \sum_{i=0}^{m-1} b_{ij} b^i$
  - Pour chaque arête j : est associé l'entier b^j

VC $\leq$ STOCKAGE (1/2)

- On se donne un graphe $G = (V, E)$ et un entier k.

  Il nous faut construire un ensemble d'entiers $\mathcal{P}$ et définir t.

- Pour cela,

  ◂ Fin

  - on numérote les sommets entre 0 et $n-1$
    et des arêtes entre 0 et $m-1$.
  - on associe un entier b_{ij} pour chaque couple (arête,sommet)

$$b_{ij} = \begin{cases} 1 & \text{si l'arête } i \text{ est incidente au sommet } j, \\ 0 & \text{sinon} \end{cases}$$

- On construit l'ensemble $\mathcal{P}$ de la façon suivante : $(b = 4)$

  - Pour chaque sommet j : $s_j = b^m + \sum_{i=0}^{m-1} b_{ij} b^i$
  - Pour chaque arête j : est associé l'entier b^j

- On construit l'entier D :

$$D = \underbrace{k b^m}_{\text{cardinalite de la couverture}} + \sum_{i=0}^{m-1} \underbrace{2 b^i}_{\text{cout de l'arete } i}$$

VC $\leq$ STOCKAGE (2/2)

- On peut prouver :

  il existe une couverture sommet du graphe G de cardinalité k

  $$\Longleftrightarrow$$

  il existe un sous-ensemble $\mathcal{P}' \subseteq \mathcal{P}$ tel que $\sum_{s \in \mathcal{P}'} s = t$

VC $\leq$ STOCKAGE (2/2)

- On peut prouver :

  il existe une couverture sommet du graphe G de cardinalité k

  $$\Longleftrightarrow$$

  il existe un sous-ensemble $\mathcal{P}' \subseteq \mathcal{P}$ tel que $\sum_{s \in \mathcal{P}'} s = t$

- La réduction se fait en temps polynomial :
  - Un entier est construit pour chaque arête et pour chaque sommet.

    Au pire des cas, il y a $\mathcal{O}(|E|)$ entiers dans l'instance.

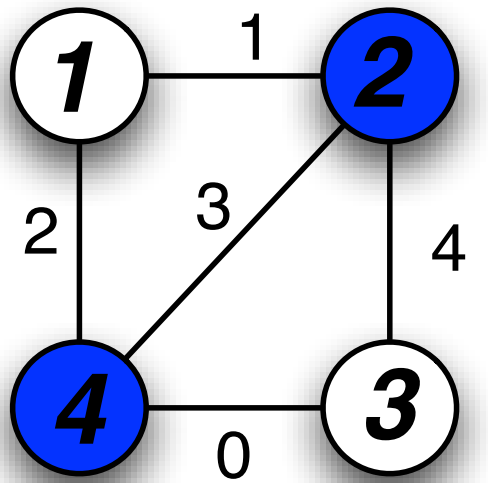
  - Chaque entier se construit en $m + 1$ opérations.

◂ Retour

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



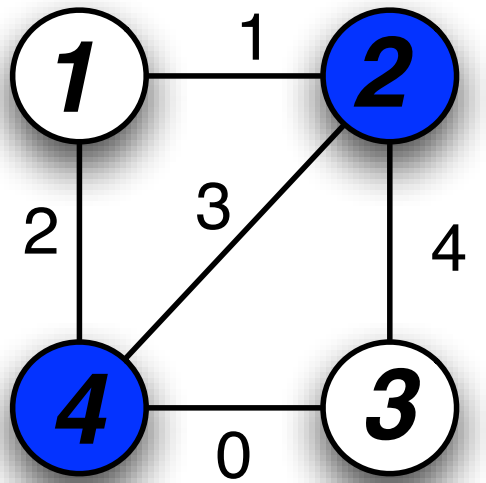
- $k = 2$

Back 1

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

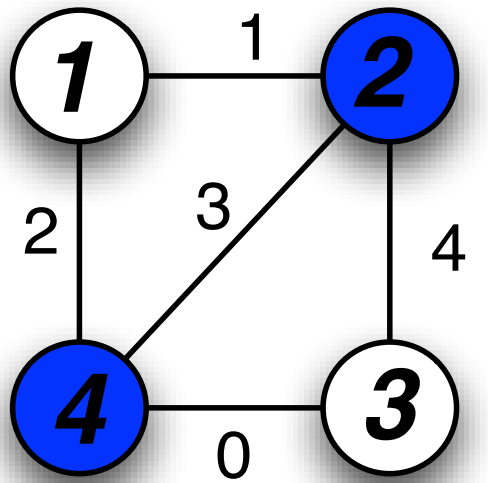
- s_1 (correspondant au sommet 1)

Back 2

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

- $s_1 = b^5$

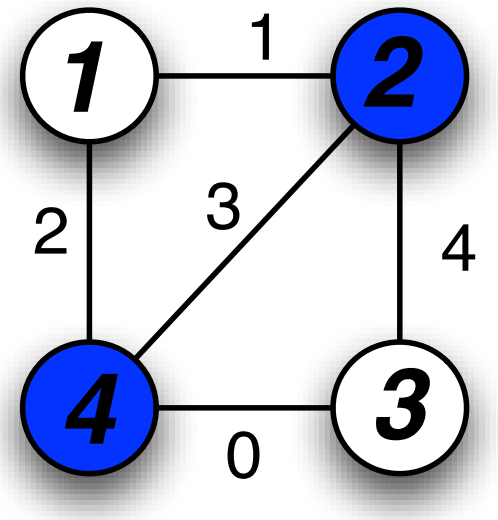
(correspondant au sommet 1)

Back 2

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

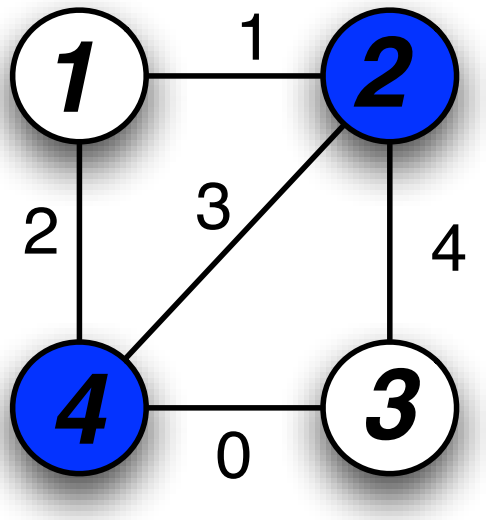
- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)

Back 2

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

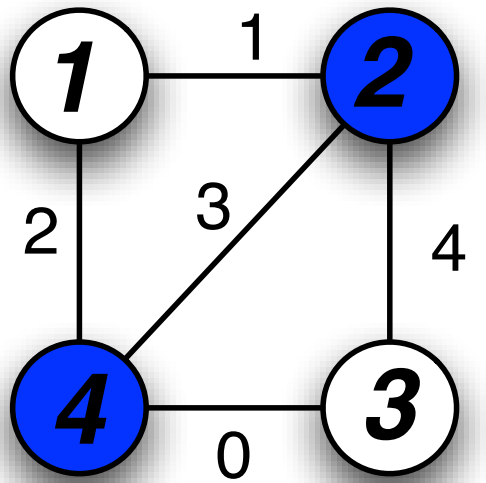
- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)
- $s_2 = b^5 + b^1 + b^3 + b^4$ (sommet 2)
- $s_3 = b^5 + b^0 + b^4$ (sommet 3)
- $s_4 = b^5 + b^0 + b^2 + b^3$ (sommet 4)

Back 2

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

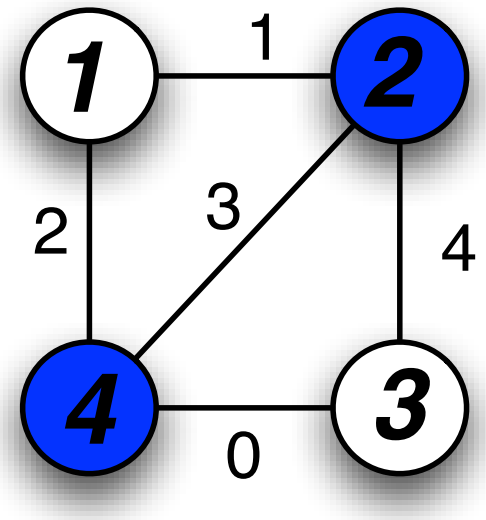
- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)
- $s_2 = b^5 + b^1 + b^3 + b^4$ (sommet 2)
- $s_3 = b^5 + b^0 + b^4$ (sommet 3)
- $s_4 = b^5 + b^0 + b^2 + b^3$ (sommet 4)
- b^0, b^1, b^2, b^3, b^4 (les entiers correspondants aux arêtes)

Back 3

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)
- $s_2 = b^5 + b^1 + b^3 + b^4$ (sommet 2)
- $s_3 = b^5 + b^0 + b^4$ (sommet 3)
- $s_4 = b^5 + b^0 + b^2 + b^3$ (sommet 4)
- b^0, b^1, b^2, b^3, b^4 (les entiers correspondants aux arêtes)

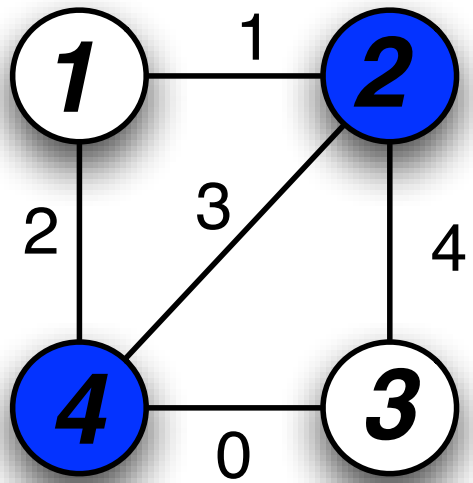
Si $S = \{2, 4\}$ est une couverture sommet, alors
$\mathcal{P}' = \{s_2, s_4, \quad\}$
Somme $= 2b^5 + b^0 + b^1 + b^2 + 2b^3 + b^4$

Back 3

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)
- $s_2 = b^5 + b^1 + b^3 + b^4$ (sommet 2)
- $s_3 = b^5 + b^0 + b^4$ (sommet 3)
- $s_4 = b^5 + b^0 + b^2 + b^3$ (sommet 4)
- b^0, b^1, b^2, b^3, b^4 (les entiers correspondants aux arêtes)

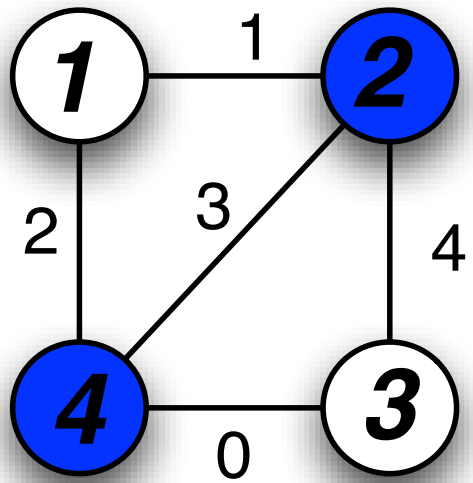
Si $S = \{2, 4\}$ est une couverture sommet, alors
$\mathcal{P}' = \{s_2, s_4, b^0, b^1, b^2, b^4\}$
Somme $= 2b^5 + 2b^0 + 2b^1 + 2b^2 + 2b^3 + 2b^4$

Back 3

Exemple

Soit $\mathcal{I} = (G, k)$ une instance de VC :

- le graphe G



- $k = 2$

L'ensemble $\mathcal{P}$ est composé des entiers suivants :

- $s_1 = b^5 + b^1 + b^2$ (correspondant au sommet 1)
- $s_2 = b^5 + b^1 + b^3 + b^4$ (sommet 2)
- $s_3 = b^5 + b^0 + b^4$ (sommet 3)
- $s_4 = b^5 + b^0 + b^2 + b^3$ (sommet 4)
- b^0, b^1, b^2, b^3, b^4 (les entiers correspondants aux arêtes)

$D = k b^5 + 2b^0 + 2b^1 + 2b^2 + 2b^3 + 2b^4$

Si $S = \{2, 4\}$ est une couverture sommet, alors
$\mathcal{P}' = \{s_2, s_4, b^0, b^1, b^2, b^4\}$
Somme $= 2b^5 + 2b^0 + 2b^1 + 2b^2 + 2b^3 + 2b^4$

Back 3