

Th: Bin packing n'est pas approximable à un facteur $\frac{3}{2} - \varepsilon$ si $\varepsilon > 0$ sauf si P = NP.

Réduction Partition \rightarrow Bin Packing (minimization)
 décision optimisation

Partition : Données : m nombres entiers a_1, a_2, \dots, a_m
 Sortie : oui si on peut répartir les ~~deux~~ entiers en deux sous ensembles de somme égale

Bin Packing : Données : m entiers b_1, b_2, \dots, b_m , $C =$
~~sous~~ : minimiser la capacité d'une boîte nécessaire pour ranger les m objets.

on part d'une instance de partition et on construit une instance de Bin Packing de la manière suivante :

$$\begin{cases} M_{BP} = M_{Part.} \\ C = \sum_{i=1}^{M_{BP}} a_i / 2 = \sum_{i=1}^{M_{BP}} b_i / 2 \\ b_i = a_i \text{ pour } i \in 1..M_{BP} = M_{Part} \end{cases}$$

* si Partition est une instance positive \Rightarrow on peut répartir en 2 ~~sous~~ ensembles de $\sum_{i=1}^{M_{BP}} a_i / 2$
 \Rightarrow Dans Bin Packing on peut ranger tous les b_i en 2 boîtes et on ne peut pas faire mieux (les 2 boîtes sont entièrement pleines.)

* si Partition est une instance négative \Rightarrow il faut 3 boîtes dans Bin Packing.

* si on a une $(\frac{3}{2} - \varepsilon)$ approx pour Bin Packing
 soit $A(I) =$ la valeur donnée par l'algorithme d'approx
 = nbre de boîtes utilisées par cet algo

$$OPT(I) \leq A(I) \leq (\frac{3}{2} - \varepsilon) OPT(I)$$

si l'instance Partition est positive $\Rightarrow OPT(I) = 2$
 négative $\Rightarrow OPT(I) = 3$

pour I positive :

$$2 \leq A(I) \leq (\frac{3}{2} - \varepsilon) 2$$

$2 \leq A(I) \leq 3 - 2\varepsilon < 3$ décide l'instance

$\Leftrightarrow A(I) = 2 \Rightarrow$ on a ~~réussi~~ ~~l'instance~~ de Partition.

Si $P \neq NP$, Partition NP-complet \Rightarrow on ne peut pas décider en temps polynomial \Rightarrow pas de $(\frac{3}{2} - \epsilon)$ -approx pour BinPacking

Gap introducing Reduction:

Π_d = problème de décision

Π_{opt} = problème d'optimisation

une réduction introduisant un écart de Π_d vers Π_{opt}

* si I une instance du problème Π_d est positive

alors $I' = R(I)$ l'instance du pb Π_{opt}

$$OPT(I') \leq f(I)^2$$

* si une instance I du problème Π_d est négative

alors $I' = R(I)$ l'instance du pb Π_{opt}

$$\text{vérifie } OPT(I') > \alpha(I) \cdot f(I)$$

Ex: binPacking

$$\begin{array}{lll} \text{Positive} \rightarrow & OPT(BP) & \leq (\frac{3}{2} - \epsilon)2 \\ \text{négative} \rightarrow & OPT(BP) & \geq (\frac{3}{2} - \epsilon)2 \end{array} \quad f(BP) = 2$$

on peut ensuite combiner une réduction qui introduit un écart avec des réductions entre problèmes d'optimisation

Gap Preserving Reduction:

Π_1, Π_2 deux problèmes d'optimisation

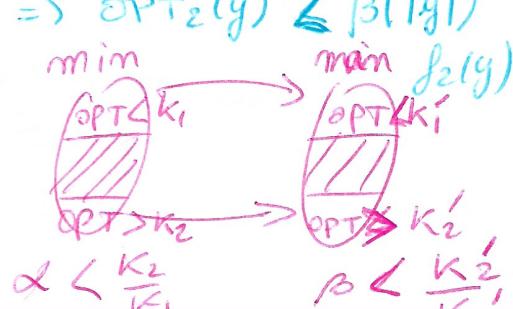
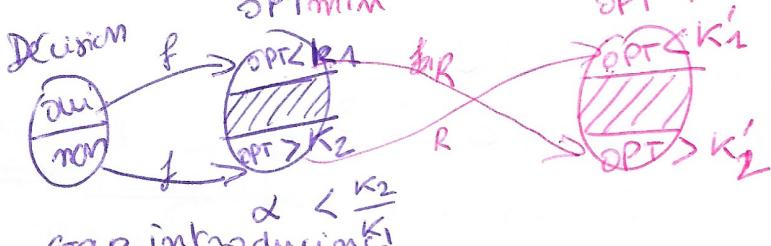
$f_1, \alpha \rightarrow \Pi_1$ minimisation

$f_2, \beta \rightarrow \Pi_2$ maximisation

soit x une instance de Π_1 , une réduction qui préserve l'écart va produire en temps polynomial en $|x|$ une instance y de Π_2 tq:

$$1) OPT_1(x) \leq f_1(x) \Rightarrow OPT_2(y) \geq f_2(x)$$

$$2) OPT_1(x) > \alpha(|x|) f_1(x) \Rightarrow OPT_2(y) < \beta(|y|)$$



Exemple: Max-3-SAT \rightarrow Max Independent Set ②

Max-3-SAT: 3SAT formule $\phi = m$ clauses

chaque clause = ou de 3 littéraux

- trouver une affectation des variables qui maximise le nombre de clauses satisfaites

ex $x_1, x_2, x_3, x_4 \rightarrow$ 8 littéraux : x_1 et \bar{x}_1 ,
 x_2 , \bar{x}_2 ,
 x_3 , \bar{x}_3 ,
 x_4 , \bar{x}_4

$$\phi = (x_1 \text{ ou } x_2 \text{ ou } \bar{x}_4) \text{ et } (\bar{x}_1 \text{ ou } x_3 \text{ ou } x_4)$$
$$\text{et } (x_2 \text{ ou } x_3 \text{ ou } \bar{x}_4) \text{ et } (x_2 \text{ ou } \bar{x}_3 \text{ ou } x_4)$$

$$\begin{cases} x_1 = \text{vrai} \\ x_2 = \text{faux} \\ x_3 = \text{vrai} \\ x_4 = \text{vrai} \end{cases}$$

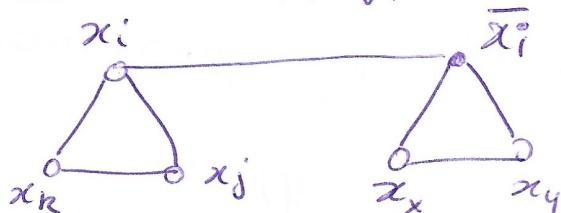
Max-Indept-Set = Graphe $G = (V, E)$

- trouver un ensemble indép. de taille maximum : dans un ensemble indépendant les sommets ne peuvent pas être voisins.
 $\forall u, v \in S \times S, (u, v) \notin E$.

S = ensemble indépendant.

Réduction :

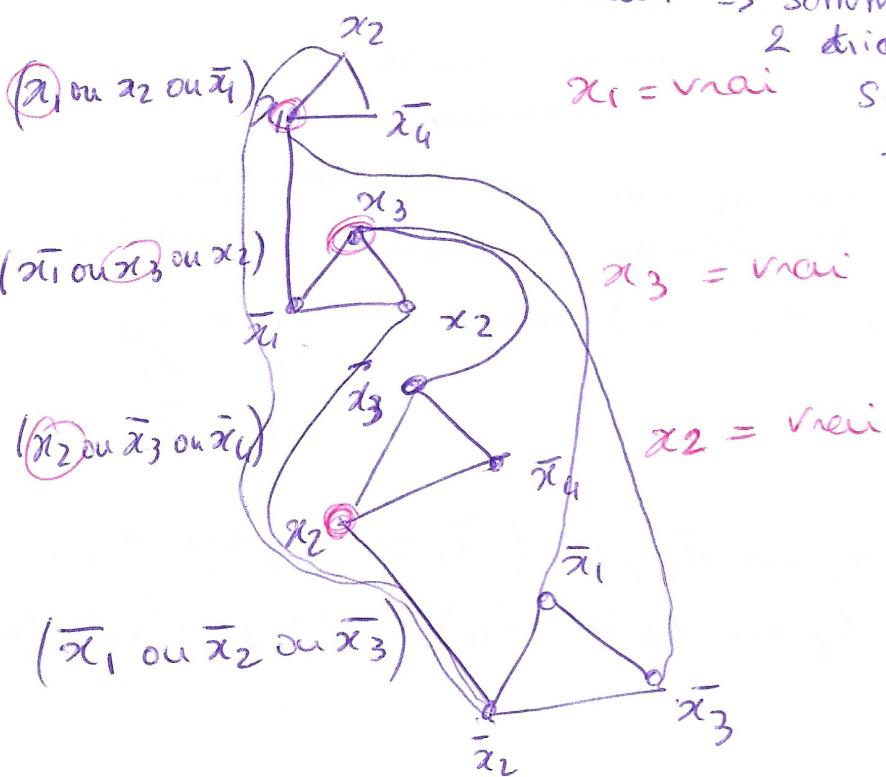
chaque clause $(x_i \text{ ou } x_j \text{ ou } x_k)$ donne un triangle dans



$(\bar{x}_i \text{ ou } x_x \text{ ou } x_y)$

on connecte les sommets représentant x_i et \bar{x}_i ($i=1 \dots n$) (n variables) par une arête

clause satisfaites \Rightarrow un littéral et à vrai dans la clause
 \Rightarrow un sommet dans le triangle correspondant
 \Rightarrow sommets indépendants : entre 2 triangles les seules arêtes sont entre x_i et \bar{x}_i
 $x_i = \text{vrai}$ sont entre x_i et \bar{x}_i
et si x_i vrai $\rightarrow \bar{x}_i$ faux
choisit \rightarrow pas choisi



(on ne choisit dans un triangle qu'un sommet correspondant à un littéral vrai qui permet de satisfaire la clause).

Inversement : on ne peut choisir qu'un sommet de chaque triangle, et pas deux sommets correspondant à même paire x_i, \bar{x}_i .

\rightarrow on met à vrai le littéral correspondant
 $\Rightarrow \emptyset$ est satisfait avec autant de clauses vraies que de sommets dans l'ensemble indépendant

Th : Max-3-SAT n'est pas approximable à un facteur $1 + \delta$ pour $\delta > 0$ sauf si $P = NP$.

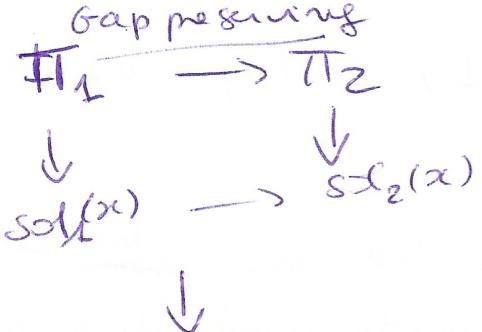
Th : Max-indep. Set n'est pas approximable à $1 + \delta$ pour $\delta > 0$.

i) $OPT_S(x) = k \rightarrow OPT_I(g) = k$

ii) $OPT_S(x) \geq f(x) = \text{nombre de clauses satisfaites} \rightarrow$

iii) $OPT_S(x) \leq (1+\delta)f(n) = (1+\delta)\text{nombre de clauses satisfaites} \rightarrow$

de PS
 * respin.
 * respin.

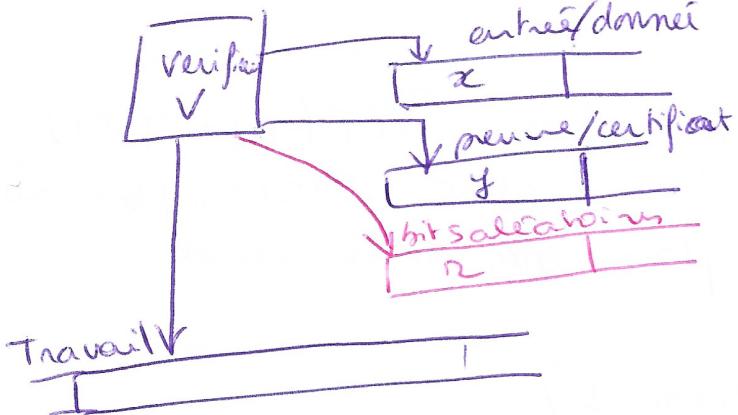


pratique pour montrer
qu'un problème n'est pas
approximable (Réduction
"légère") mais ne garantit
pas la transmission des
appartenances aux classes
de complexité.

$$\begin{array}{c} AP\text{-réduction}, C\text{-réduction} \\ \text{et autres.} \\ \#T_1 \rightarrow \#T_2 \\ x \xrightarrow{f} f(x) \\ \downarrow \\ g(x,y) \leftarrow f \quad y \in sol(f(x)) \\ \in sol(x) \end{array}$$

plutôt pour montrer
que deux problèmes sont
dans la même classe (ou
complet pour la classe)
→ peuvent être + utiles
à mettre en œuvre.

PCP



NP : certificat
→ preuve
o algo de vérification
Det. - polynomial.

Vérifieur = machine de Turing déterministe polynomial
qui dispose en plus de bits aléatoires.

fondement :

- la machine/vérifieur peut lire à l'impoète quel bit de la preuve simplement en donnant son adresse
- les bits de la preuve lus dépendent de l'entrée et de la chaîne aléatoire
- après calcul et consultation de bits de l'entrée le vérifieur accepte ou rejette la preuve : preuve valide ou pas valide.

un langage $L \in \text{PCP}(\log n, 1)$ s'il y a un vérificateur V et une constante q

tq sur l'entrée x , V utilise une chaîne aléatoire r de taille $c \log|x|$ et regarde q bits de la preuve.

et :

- Si $x \in L$, alors \exists une preuve y tq V accepte avec probabilité 1 \rightarrow peu importe la chaîne aléatoire r de taille $c \log|x|$ donnée.

- Si $x \notin L$ alors ~~si~~ il y a une preuve y que V accepte y avec probabilité $\leq 1/2$
(sur toutes les 2 possibles de taille $c \log|x|$)

Probabilité d'enrou = probabilité d'accepter y preuve non valide.

pour $r(n)$ et $q(n)$ on définit $\text{PCP}(r(n), q(n))$
= classe tq le vérificateur utilise $O(r(n))$ bits aléatoires
et regarde $O(q(n))$ bits de la preuve.

Th : $\text{NP} = \text{PCP}(\log n, 1)$

on a $O(\log n)$ bits aléatoires
on regarde $O(1) =$ notre constante de bits de la preuve.