# Le problème du Bin Packing (remplissage de sacs)

Johanne Cohen

Johanne.Cohen@loria.fr

Laboratoire LORIA

# Objectif de la présentation

METTRE un ensemble d'éléments fixés dans des sacs de même
capacité en utilisant le moins possible de sacs.

# Plan

1. Présentation du problème.
2. Algorithme glouton d'approximation.
3. Inapproximation.
4. Remarque sur le codage des données.

# Petit rappel sur la compléxité:

**Définition:**

Un problème de décision $\Pi = (D_\Pi, Y_\Pi)$ correspond

à un ensemble d'instance $D_\Pi$

et à un sous-ensemble $Y_\Pi \subset D_\Pi$ d'instances positives.

REMARQUE:

Problème de décision $\neq$ problème de construction.

# Énoncé du problème Bin Packing:

INSTANCE:

un ensemble $\mathcal{B}$ de boites

une fonction taille $w : \mathcal{B} \longrightarrow \mathbb{N}$

un entier $c$

un entier $k$

QUESTION:

Existe-t-il un rangement $\mathcal{R}$ des boites de $\mathcal{B}$

dans $k$ sacs de capacité $c$?

**Remarque:** Un rangement $\mathcal{R}$ est une application $\mathcal{B} \longrightarrow [1...k]$ tel que

1. $\mathcal{R}(b) = j$ signifie que la boite $b$ est placée dans le sac $j$.
2. $\forall j \in [1...k], \sum_{b \in \mathcal{B} | \mathcal{R}(b) = j} w(b) \leq c$

# Definition de la classe NP

**Définition:** Une relation binaire $R(x,y)$ est polynomiale

si il existe un polynome $Poly$ tel que étant donnée $x$ et $y$, on peut determiner si $R(x,y)=1$ en temps $Poly(|x|)$

**Définition:** Un problème de décision $\Pi = (D_\Pi, Y_\Pi)$ est dans $NP$ si et seulement si il existe une relation binaire $R(x,y)$ est polynomiale tel que

$\forall x \in D_\Pi, x \in Y_\Pi$ si et seulement si $\exists y$ tel que $R(x,y)=1$

# Exemple d'un problème dans $NP$.

Le problème Bin packing est dans NP car:

$x =$ une instance de Bin packing

$y =$ un rangement.

$R(x, y)$: $D_\Pi \times ens\_de\_rangement \to \{0, 1\}$ vérifie en temps polynomiale que le rangement $y$ est correct par rapport à l'ínstance $x$.

*Rappel: Un problème de décision $\Pi = (D_\Pi, Y_\Pi)$ est dans $NP$ si et seulement si il existe une relation binaire $R(x, y)$ est polynomiale tel que*

$$\forall x \in D_\Pi, x \in Y_\Pi \text{ si et seulement si } \exists y \text{ tel que } R(x, y) = 1$$

# Définition: Transformation polynômiale

Un problème $Pb1$ est plus simple qu'un problème $Pb2$ (noté $Pb1 \leq Pb2$) : s'il existe une fonction $f : D_{Pb1} \longrightarrow D_{Pb2}$ qui se calcule en temps polynômial telle que

$$x \in Y_{Pb1} \;\; ssi \; f(x) \in \; Y_{Pb2}$$

# Résultats à venir.

1. Le problème du bin Packing est NP-complet.
2. Un algorithme glouton est une $2$-approximation.
3. Il n'existe pas d'algorithmes polynômiaux. d'approximation pour le problème Bin Packing ayant un rapport de $3/2 - \epsilon$.

# Complexité

**Théorème 1:** Le problème du Bin Packing est NP-complet.

**Preuve**:

1. Problème est dans NP.
2. Réduction avec le problème Partition:

INSTANCE:

un ensemble $S = \{s_1, \ldots, s_m\}$

une fonction de poids $\mathcal{W} : S \to \mathbb{N}$

QUESTION:

Existe-t-il une partition de $S$ en $2$ sous-ens $S_1$ et $S_2$

$\sum_{s \in S_1} \mathcal{W}(s) = \sum_{s \in S_2} \mathcal{W}(s)$

$\square$

# Complexité

**Théorème 1:** Le problème du bin Packing est NP-complet.

**Preuve**:

1. Problème est dans NP.
2. Réduction avec le problème Partition:

INSTANCE:

un ensemble $S = \{s_1, \ldots, s_m\}$

une fonction de poids $\mathcal{W} : S \to \mathbb{N}$

QUESTION:

Existe-t-il une partition de $S$ en $2$ sous-ens $S_1$ et $S_2$

$\sum_{s \in S_1} \mathcal{W}(s) = \sum_{s \in S_2} \mathcal{W}(s)$

$\square$

# Complexité

**Théorème 1:** Le problème du bin Packing est NP-complet.

**Preuve**:

1. Problème est dans NP.
2. Réduction avec le problème Partition:
   Définissons la transformation polynômial $\mathcal{F}$

$$\mathcal{F}(S,\mathcal{W}) = \begin{array}{l} \mathcal{B} = S \\ \forall s \in S, w(s) = \mathcal{W}(s) \\ c = \frac{\sum_{s\in S}\mathcal{W}(s)}{2} \\ k = 2 \end{array}$$

$\square$

# Illustration de la transformation

Partition

$S = \{a_1, a_2, a_3, a_4, a_5\}$

$\mathcal{W}(a_1, a_2, a_3, a_4, a_5) = \{6, 7, 2, 4, 1\}$

$\longrightarrow$

Bin Packing

$\mathcal{B} = \{a_1, a_2, a_3, a_4, a_5\}$

$w\ (a_1, a_2, a_3, a_4, a_5) = \{6, 7, 2, 4,$

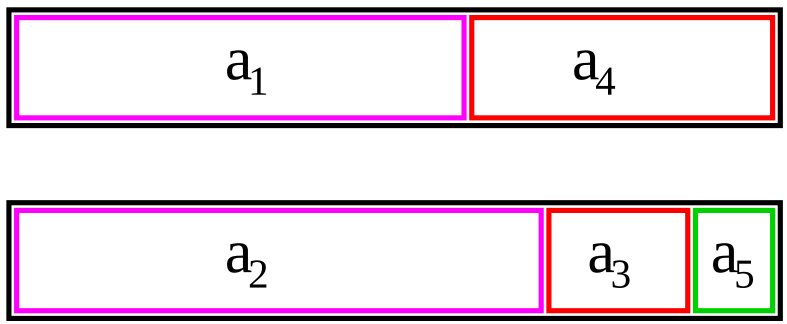
$c = 10,\ k = 2$

Réponse: oui car

$S_1 = \{a_1, a_4\}$

$S_2 = \{a_2, a_3, a_5\}$

Réponse: oui car



# Plan

1. Présentation du problème.

2. Algorithme glouton d'approximation.

3. Inapproximation.

4. Remarque sur le codage des données.

# Comment résoudre un pb. NP-complet?

Difficulté à trouver un algorithme polynomiale si le problème est NP-complet.

Existance de $2$ approches de résolutions: peut convenir

1. un algo. exponentiel si la taille de l'ínstance est petite.
2. un algo. polynomial trouvant une solution *"presque"* optimale.

# Petit rappel sur la compléxité:

Un problème d'optimisation combinatoire de minimisation $\Pi$ est défini de la facon suivante:

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

  Pour chaque instance $I$ de $D_\Pi$, et pour chaque candidat $\sigma \in S_\Pi(I)$ :

  $m_\Pi(I, \sigma)$ est la valeur de la solution (un rationnel).

# Petit rappel sur la compléxité:

Si le problème $\Pi$ est un problème de minimisation, alors la solution optimale pour l'instance $I$ de $D_\Pi$ est la ou une solution $\sigma^* \in S_\Pi(I)$ tel que

$$\forall \sigma \in S_\Pi(I),\ m_\Pi(I, \sigma^*) \leq m_\Pi(I, \sigma)$$

Notons $OPT_\Pi(I) = m_\Pi(I, \sigma^*)$

Un algorithme $A$ est un algorithme d'optimisation de $\Pi$ si pour toute instance $I$ de $D_\Pi$, $A$ retourne un $\sigma \in S_\Pi(I)$ optimal.
Notons dans ce cas $A(I) = m_\Pi(I, \sigma)$ pour le $\sigma$ retourné.

# Rapport d'approximation.

Soit $\Pi$ un prob. de minimisation. Soit $I$ une instance de $D_\Pi$

Le rapport d'un algorithme $A$ est $R_A(I) = \frac{A(I)}{OPT_\Pi(I)}$.

Le rapport d'approximation absolu $R_A$ pour un algorithme d'approximation est défini:

$$R_A = inf\{r \geq 1 : R_A(I) \leq r,\ \forall I \in D_\Pi\}$$

# L'algorithme Next Fit.

ENTRÉE: un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,

une fonction $w : \mathcal{B} \to \mathbb{N}$, un entier $c$

SORTIE: un entier $k$

ALGORITHME:

1. $j = 1$
2. Pour $i$ allant de $1$ à $n$ faire
   (a) Si $b_i$ peut être mis dans $S_j$, $S_j \leftarrow S_j \cup \{b_i\}$
   (b) Sinon
      i. $j \leftarrow j + 1$
      ii. $S_j \leftarrow \{b_i\}$
3. retourner $j$

# Illustration de l'algo. Next Fit.

INSTANCE:

deux entiers $\alpha$, et $n$ tel que $n = 4\alpha$ ,

un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,

une fonction $w$ tel que $w(\mathcal{B}) = \{\alpha, 1, \alpha, 1, \ldots, \alpha, 1\}$

$c = 2\alpha$

# Illustration de l'algo. Next Fit.

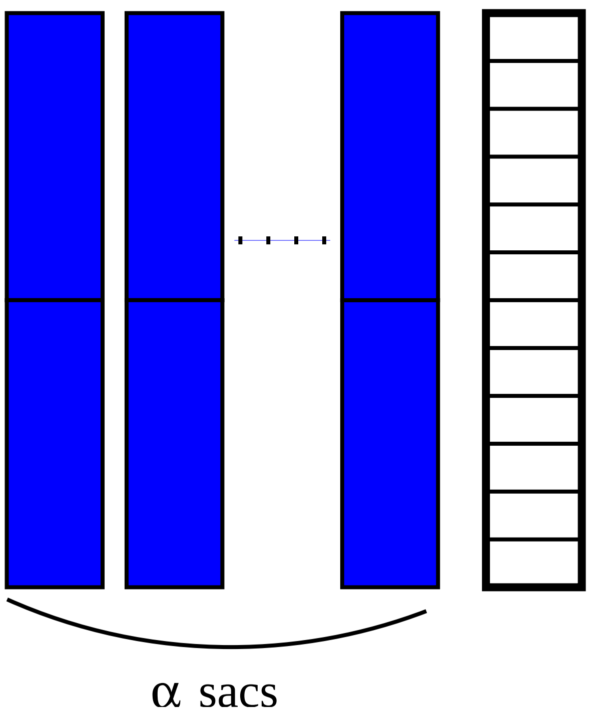
INSTANCE: deux entiers $\alpha$, et $n$ tel que $n = 4\alpha$ ,

un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,

une fonction $w$ tel que $w(\mathcal{B}) = \{\alpha, 1, \alpha, 1, \ldots, \alpha, 1\}$

$c = 2\alpha$

SOLUTION OPTIMALE: $\alpha + 1$



# Illustration de l'algo. Next Fit.

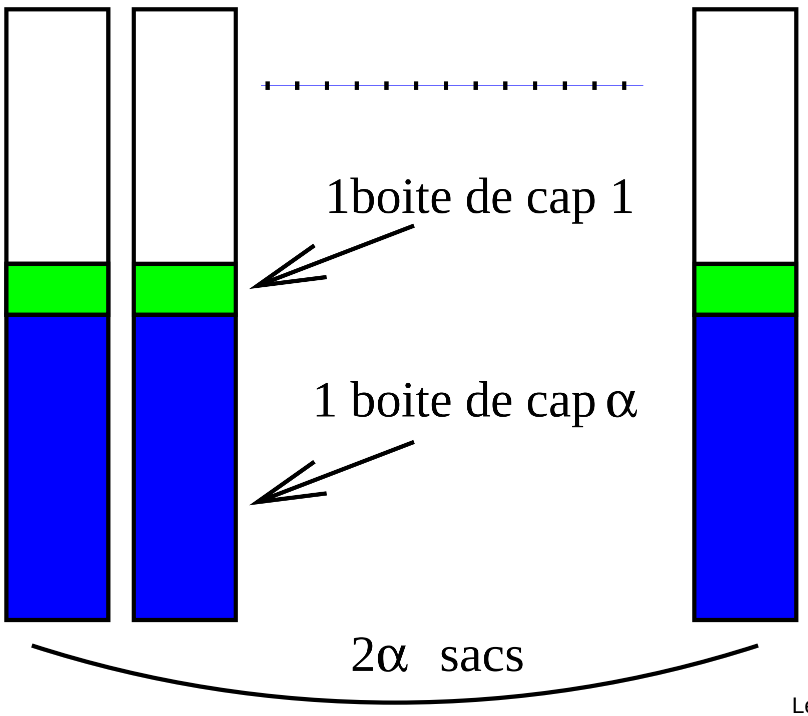
INSTANCE: deux entiers $\alpha$, et $n$ tel que $n = 4\alpha$ ,

un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,

une fonction $w$ tel que $w(\mathcal{B}) = \{\alpha, 1, \alpha, 1, \ldots, \alpha, 1\}$

$c = 2\alpha$

SOLUTION OPTIMALE: $\alpha + 1$ ET SOLUTION FOURNI PAR L'ALGO: $2\alpha$



# acteur d'approximation de l'algo. Next Fit

**Théorème 2:** L'algorithme Next Fit retourne une solution à un facteur $2$ de l'optimal.

**Preuve**:

- Notons $\beta = \sum_{b \in \mathcal{B}} w(b)$ et $k$ la valeur retournée par l'algo.
- $OPT \geq \beta / c$
- Considérons $i$ tel que $b_i \in S_j$ et $b_{i+1} \in S_{j+1}$
  - il existe $\alpha$ tel que $\alpha \leq \ell \leq i, b_\ell \in S_j$
  - $\sum_{\ell=\alpha}^{i} w(b) \leq c$ et $c < \sum_{\ell=\alpha}^{i+1} w(b) \leq 2c$
- Globalement, $kc < 2 \sum_{\ell=\alpha}^{i+1} w(b)$ donc $k \leq 2\beta/c \leq 2OPT$

□

# L'algorithme First Fit Decreasing.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \rightarrow \mathbb{N}$

un entier $c$

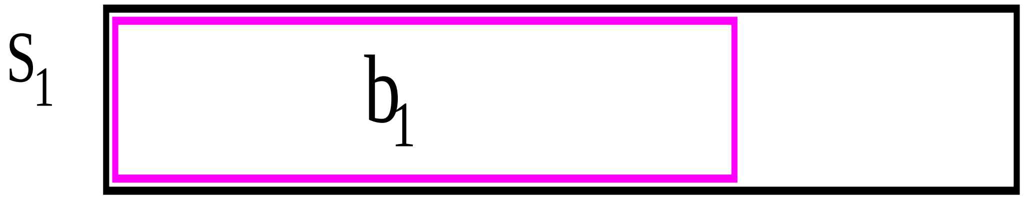
SORTIE: un entier $k$

ALGORITHME:

1. Trier $\mathcal{B} = \{b_1, \ldots, b_n\}$ dans l'ordre décroissant en fct de $w$
2. $k := 1$ , $S_1 := \emptyset$
3. Pour $i$ allant de $1$ à $n$ faire
   (a) Si $b_i$ peut être mis dans un sac $S_j$ avec $j \leq k$, insérer
   (b) Sinon
      i. créer un nouveau sac $S_{k+1}$ ($S_{k+1} := \emptyset$)
      ii. insérer $b_i$ dans $S_{k+1}$ et réactualiser $k$ ($k := k + 1$)
4. Retourner $k$

# Illustration de l'algorithme

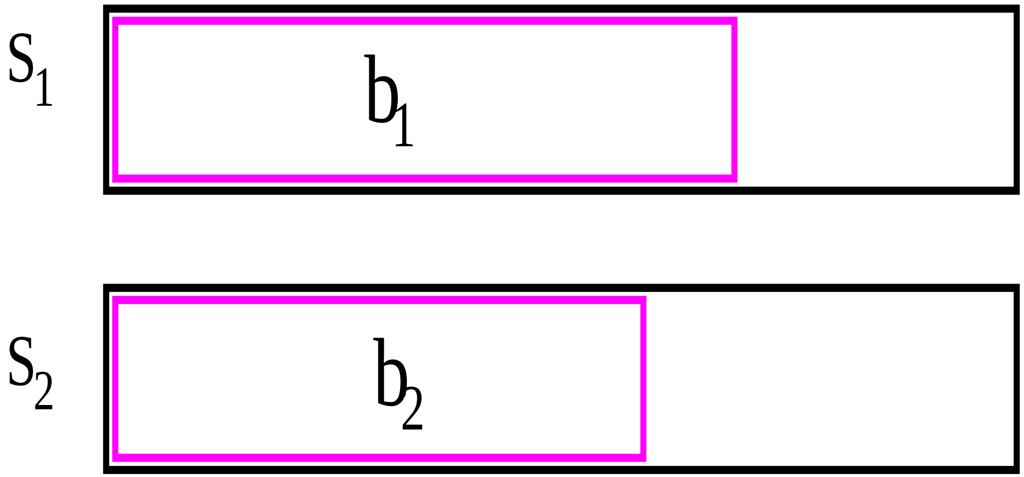
EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.




k=1

# Illustration de l'algorithme

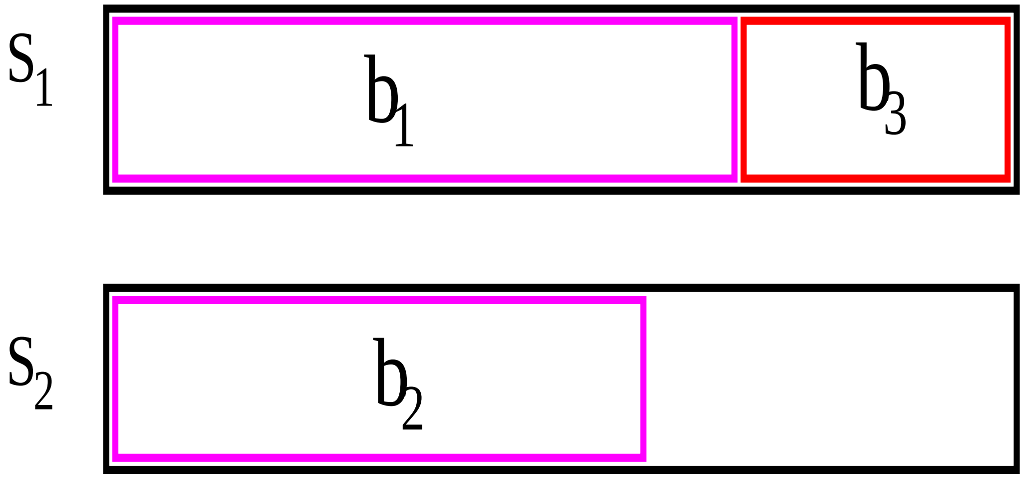
EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.




k=1

# Illustration de l'algorithme

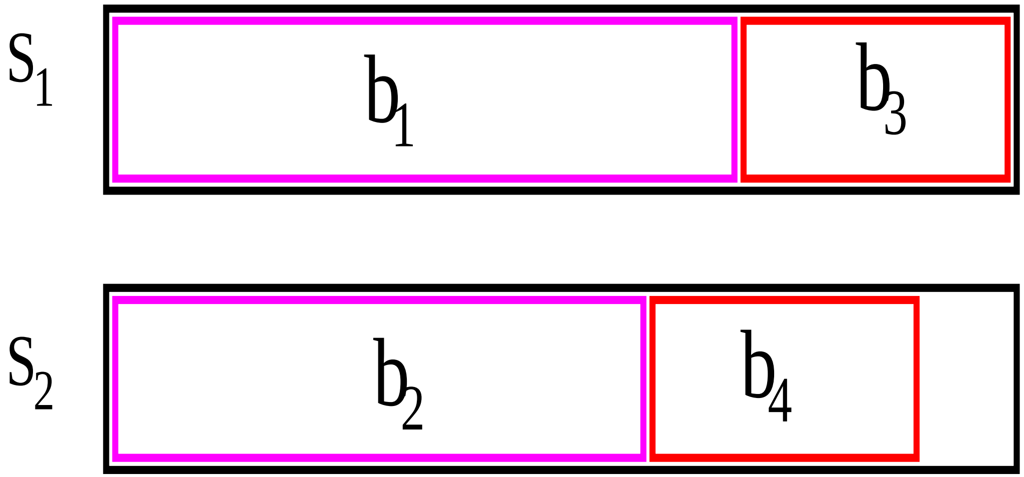
EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.




k=2

# Illustration de l'algorithme

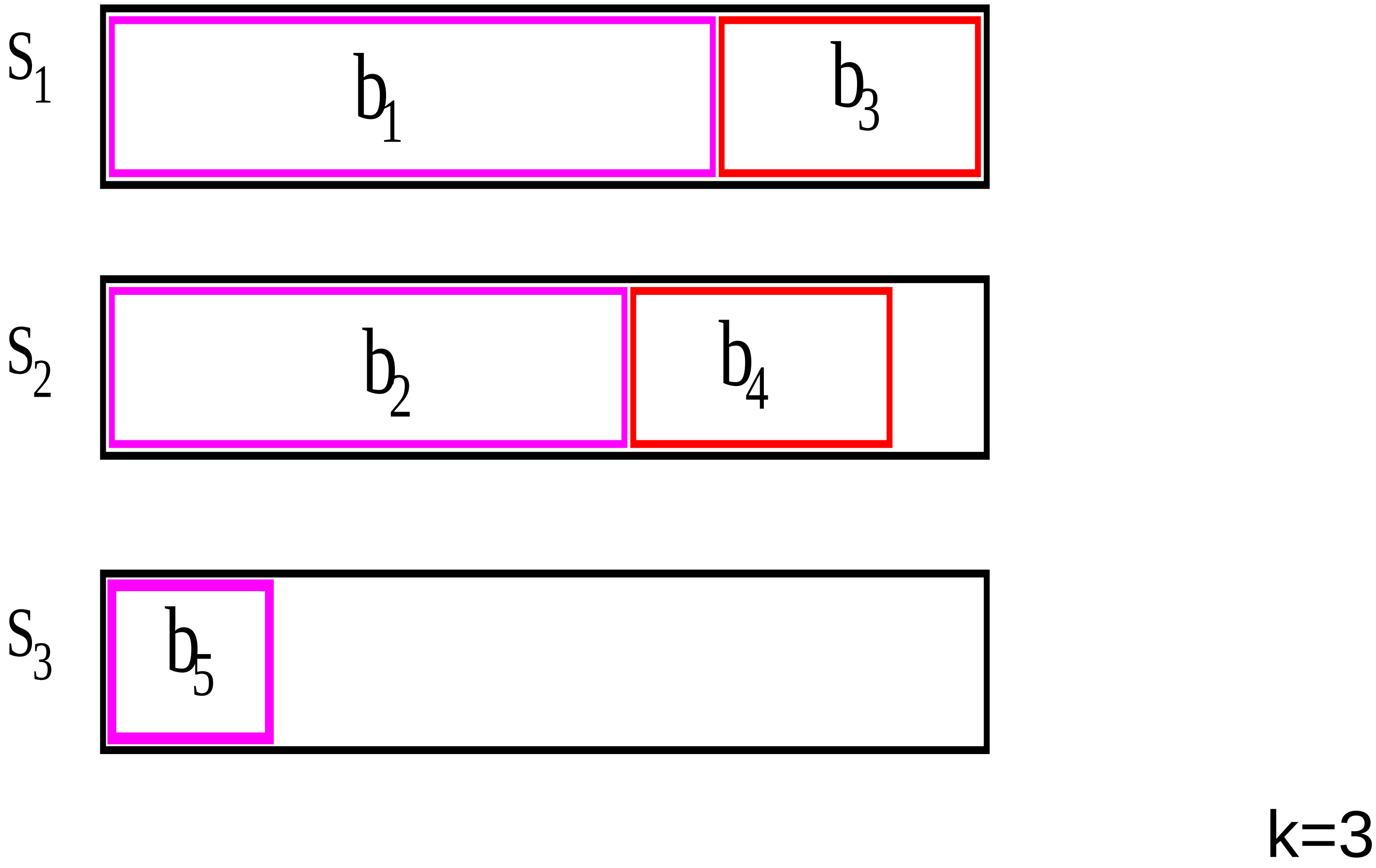
EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.




k=2

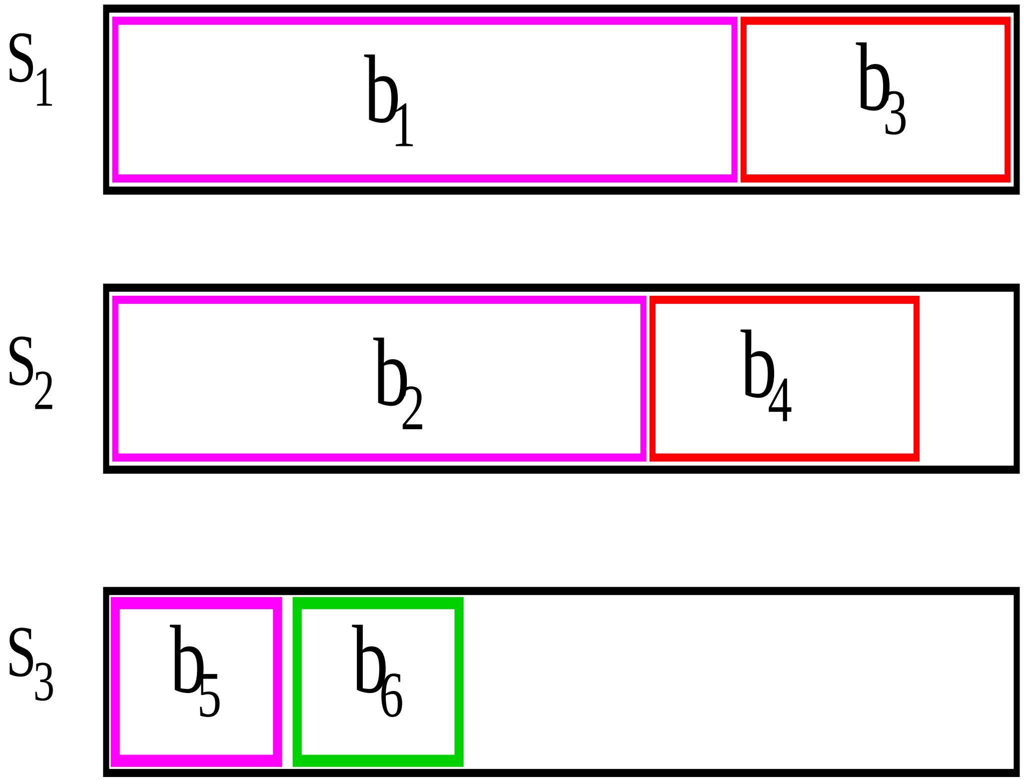
# Illustration de l'algorithme

EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



# Illustration de l'algorithme

EXEMPLE: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.




k=3

# Facteur d'approximation

**Théorème 3:** L'algorithme First Fit Decreasing retourne une solution à un facteur $2$ de l'optimal.

REMARQUE

**Proposition 1:** $OPT \geq (\sum_{i=1}^{n} w(b_i))/c$.

Car la borne inférieure correspond à une solution où tous les sacs sont pleins.

# Facteur d’approximation

**Théorème 3:** L’algorithme First Fit Decreasing retourne une solution à un facteur $2$ de l’optimal.

**Preuve**:

CAS 1 OÙ $w(b_n) \geq c/2$: l’algorithme glouton est optimal.


1 élément par sac.

EXEMPLE: $c = 10, w(b_1) = 8$, $w(b_2) = 7$, $w(b_3) = 5$, $w(b_4) = 5$

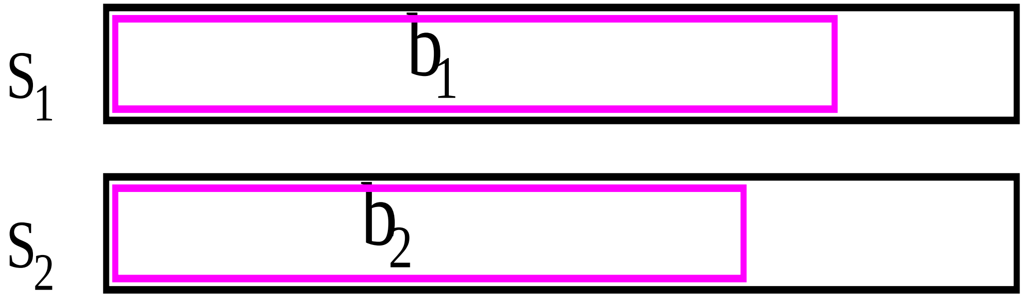
$\square$

# Facteur d'approximation

**Théorème 3:** L'algorithme <u>First Fit Decreasing</u> retourne une solution à un facteur $2$ de l'optimal.

**Preuve**:

CAS 1 OÙ $w(b_n) \geq c/2$: l'algorithme glouton est optimal.




1 élément par sac.

EXEMPLE: $c = 10, w(b_1) = 8$, $w(b_2) = 7$, $w(b_3) = 5$, $w(b_4) = 5$

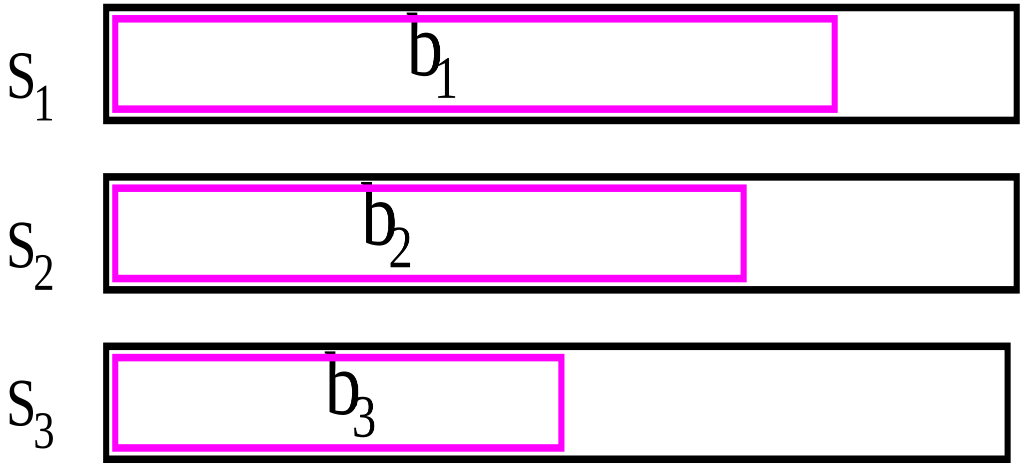
□

# Facteur d'approximation

**Théorème 3:** L'algorithme First Fit Decreasing retourne une solution à un facteur $2$ de l'optimal.

**Preuve**:

CAS 1 OÙ $w(b_n) \geq c/2$: l'algorithme glouton est optimal.




Au plus 2 éléments par sac.

EXEMPLE: $c = 10, w(b_1) = 8$, $w(b_2) = 7$, $w(b_3) = 5$, $w(b_4) = 5$

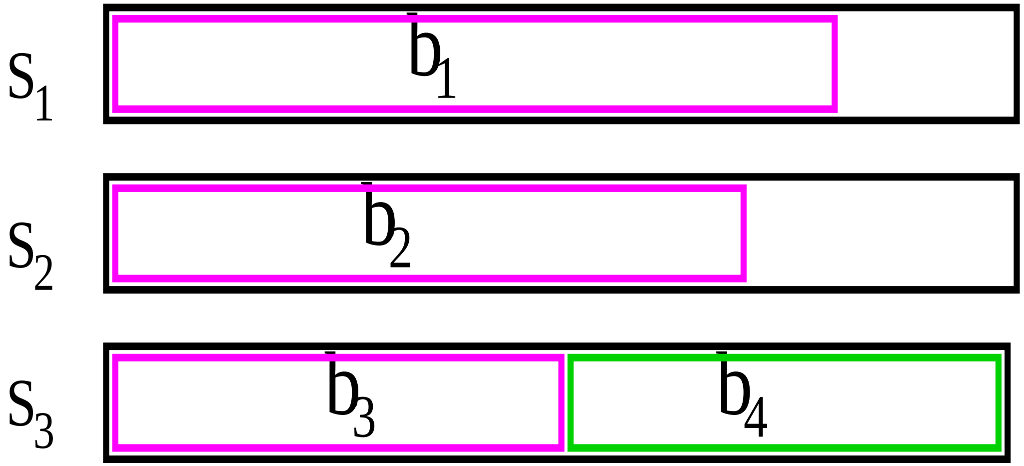
$\square$

# Facteur d'approximation

**Théorème 3:** L'algorithme First Fit Decreasing retourne une solution à un facteur $2$ de l'optimal.

**Preuve**:

CAS 1 OÙ $w(b_n) \geq c/2$: l'algorithme glouton est optimal.




Au plus 2 éléments par sac.

EXEMPLE: $c = 10, w(b_1) = 8$, $w(b_2) = 7$, $w(b_3) = 5$, $w(b_4) = 5$

$\square$

# Suite de la preuve.

**Théorème 3:** L'algorithme First Fit Decreasing retourne une solution à un facteur $2$ de l'optimal.

CAS 2 OÙ $w(b_n) < c/2$:

Soit $b_j$ la boite pour laquelle le sac $k$ a été crée.

1. capacité des $k-1$ premiers sacs remplis ( = $c(k-1)$)

   = place vide ($\mathcal{P}$) + place occupée ($\mathcal{O}$)

   (a) $\mathcal{P} < (k-1)w(b_j)$

   (b) $\mathcal{O} < \sum_{i=1, i\neq j}^{n} w(b_i)$

2. $c(k-1) < \sum_{i=1}^{n} w(b_i) + w(b_j)(k-1) < OPTc + (k-1)c/2$

   Considèrons uniquement le cas où $(w(b_j) < c/2)$

3. $k - 1 < 2OPT \rightarrow k \leq 2OPT$

# Considérons l'exemple

deux entiers $\alpha$, et $n$ tel que $n = 5\alpha$ ,

un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,

une fonction $w$ tel que $\forall i, 1 \leq i\alpha w(b_i) = 1/2 + \epsilon$,

INSTANCE:

$\forall i, 1 \leq i \leq \alpha, w(b_i) = 1/2 + \epsilon$,

$\forall i, \alpha + 1 \leq i \leq 2\alpha, w(b_i) = 1/4 + 2\epsilon$,

$\forall i, 2\alpha + 1 \leq i \leq 3\alpha, w(b_i) = 1/4 + \epsilon$,

$\forall i, 3\alpha + 1 \leq i \leq 5\alpha, w(b_i) = 1/4 - 2\epsilon$,

$c = 2\alpha$

SOLUTION OPTIMALE: $= \frac{3}{2}\alpha$

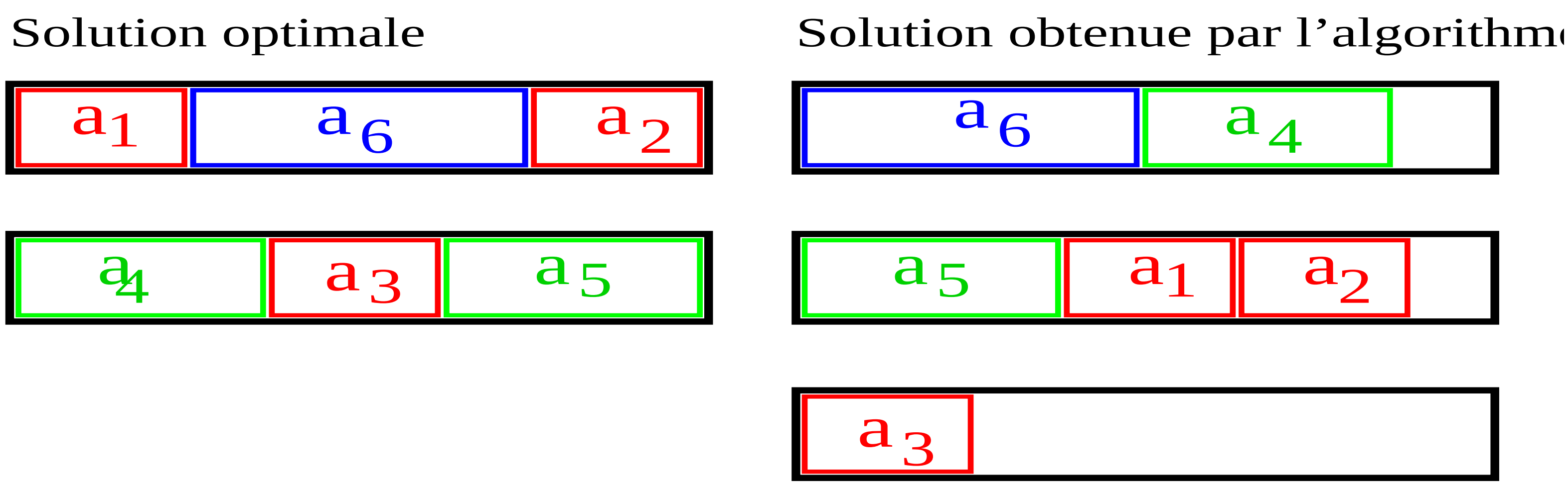
SOLUTION FOURNI PAR L'ALGO: $= \frac{11}{6}\alpha$

# Plusieurs questions se posent alors. . .

1. Peut-on améliorer la borne inférieure de la prop.1?
2. Peut-on mieux évaluer le rapport d'approximation de l'algorithme First Fit Decreasing ?
3. Peut-on trouver un algorithme ayant un meilleur rapport d'approx.?

# Plusieurs questions se posent alors…

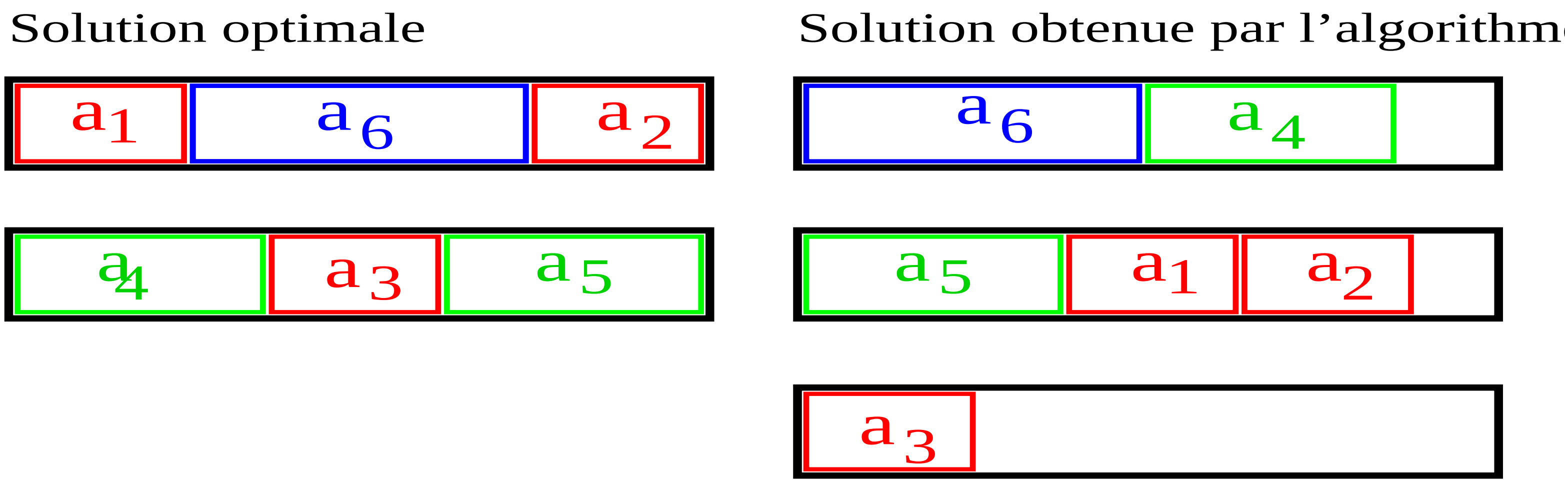
1. Peut-on améliorer la borne inférieure de la prop.1?non
2. Peut-on mieux évaluer le rapport d'approximation de l'algorithme First Fit Decreasing ?
3. Peut-on trouver un algorithme ayant un meilleur rapport d'approx.?




$$c = 8, w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,$$
$$w(a_6) = 4$$

# Plusieurs questions se posent alors. . .

1. Peut-on améliorer la borne inférieure de la prop.1?non
2. Peut-on mieux évaluer le rapport d'approximation de l'algorithme First Fit Decreasing ?peut-être mais il est au moins égal à $3/2$.
3. Peut-on trouver un algorithme ayant un meilleur rapport d'approx.?




$c = 8, w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,$

# Plan

1. Présentation du problème.

2. Algorithme glouton d'approximation.

3. Inapproximation.

4. Remarque sur le codage des données.

# Définition de gap-introducting reduction.

Soit $\Pi$ un problème de minimisation.

Une gap-introducting réduction à partir de 3-SAT vers $\Pi$ corresponds à $2$ paramètres $f$ et $\alpha$. La réduction transforme une instance de 3-SAT, $\phi$ en une de $\Pi$ noté $x$ telle que

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

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

REMARQUE: La réduction montre qu' il n'existe pas d'algorithme d'approximation de rapport $\alpha(|x|)$ si $P \neq NP$

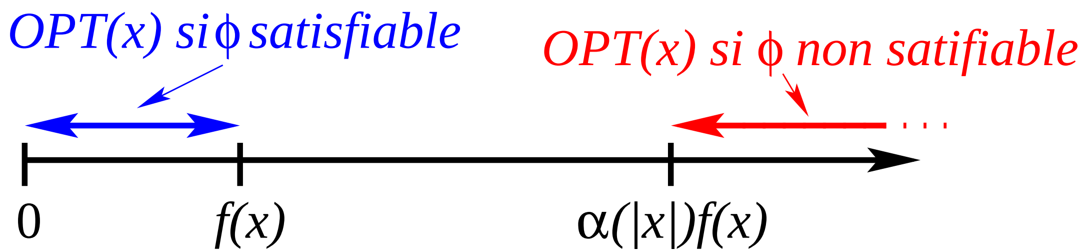
# Explication de la remarque.

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

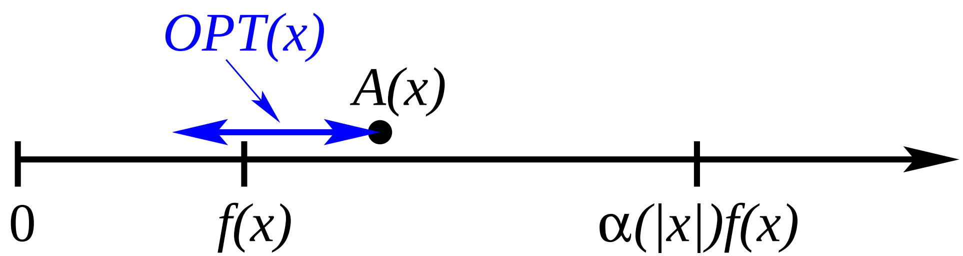
1. Soit $A$ un algo. poly. d'approx. de rapport $r \leq \alpha(|x|)$ pour $\Pi$.
2. Construisons un algorithme $\mathcal{A}'$ pour 3-SAT
   (a) $x \leftarrow \mathcal{R}(\phi)$
   (b) $a(x) \leftarrow A(x)$
   (c) Si $a(x) \leq \alpha(|x|)f(x)$, retourner VRAI
   (d) Sinon $a(x) > \alpha(|x|)f(x)$ retourner FAUX .
3. Algo. $\mathcal{A}'$polynomial $\Longrightarrow$ contradiction avec $P \neq NP$

# Illustration

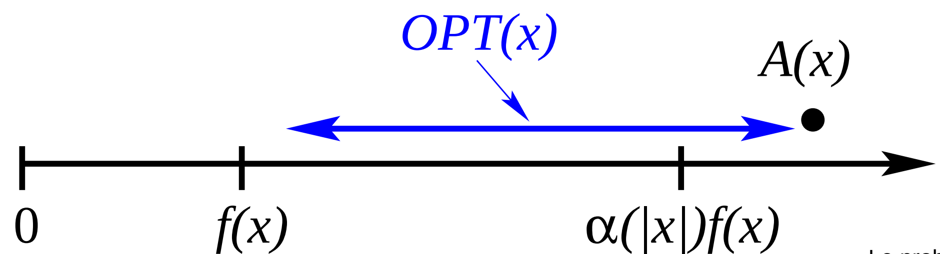
Par réduction,




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




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



# Définition du problème Partition.

INSTANCE:

un ensemble $S = \{s_1, \ldots, s_m\}$

une fonction de poids $\mathcal{W} : S \rightarrow \mathbb{N}$

QUESTION:

Existe-t-il une partition de $S$ en $2$ sous-ens $S_1$ et $S_2$

$\sum_{s \in S_1} \mathcal{W}(s) = \sum_{s \in S_2} \mathcal{W}(s)$

# Transformation

OBJECTIF: Transformer $1$ instance de Partition en $1$ instance du Bin Packing.

Transformation $\mathcal{F}(S, \mathcal{W}) = \begin{cases} \mathcal{B} = S \\ \forall s \in S, w(s) = \mathcal{W}(s) \\ c = \frac{\sum_{s \in S} \mathcal{W}(s)}{2} \end{cases}$

**Proposition 1:** Soit $\mathcal{I}$ une instance du problème partition. Soit $\mathcal{I}' = \mathcal{F}(\mathcal{I})$ (instance du problème bin packing).

la solution de $\mathcal{I}'$ $\begin{cases} = 2 \text{ si } \mathcal{I} = (S, \mathcal{W}) \text{ possède une partition,} \\ \geq 3 \text{ sinon} \end{cases}$

# Preuve

$$\text{Transformation } \mathcal{F}(S, \mathcal{W}) = \begin{cases} \mathcal{B} = S \\ \forall s \in S, w(s) = \mathcal{W}(s) \\ c = \frac{\sum_{s \in S} \mathcal{W}(s)}{2} \end{cases}$$

**Proposition 1:** Soit $\mathcal{I}$ une instance du problème partition. Soit $\mathcal{I}' = \mathcal{F}(\mathcal{I})$ (instance du problème bin packing).

la solution de $\mathcal{I}'$ $= 2$ si $\mathcal{I} = (S, \mathcal{W})$ possède une partition,
$\geq 3$ sinon

**Preuve**:

1. Si $S$ peut se partitionner équitable en $2$ sous ensembles, alors dans $\mathcal{I}'$, 1 sac = 1 partition.
2. Sinon, il faut au moins $3$ sacs.

# Résultat d'inapproximabilité.

**Théorème 3:** Si $P \neq NP$, il n'existe pas d'algorithme polynomiale d'approximation pour le problème Bin Packing ayant un rapport inferieure à $3/2 - \epsilon$.

**Preuve**: par l'absurde.

Soit $\mathcal{A}$ un algo. poly. d'approx. pour Bin Packing ayant un rapport $\alpha \leq 3/2 - \epsilon$.

Construisons un algorithme $\mathcal{A}'$ qui résoud le problème partition à partir de $\mathcal{A}$.

...

$\square$

# Algorithme $\mathcal{A}'$.

ENTRÉE: un ens. S, une fonction $\mathcal{W}$

SORTIE: un booleen

ALGORITHME:

1. $\mathcal{I}' = \mathcal{F}(\mathcal{I})$
2. Appliquer l'algo. $\mathcal{A}$ sur l'instance $\mathcal{I}'$.
3. Si le nombre de sacs est $\geq 3$, retourner faux,
4. sinon, retourner vrai.

Complexité: ?

# Algorithme $\mathcal{A}'$.

ENTRÉE: un ens. S, une fonction $\mathcal{W}$

SORTIE: un booleen

ALGORITHME:

1. $\mathcal{I}' = \mathcal{F}(\mathcal{I})$
2. Appliquer l'algo. $\mathcal{A}$ sur l'instance $\mathcal{I}'$.
3. Si le nombre de sacs est $\geq 3$, retourner faux,
4. sinon, retourner vrai.

Complexité: dépendant de la complexité de $\mathcal{A}$ et de $\mathcal{F}$

# Algorithme $\mathcal{A}'$.

ENTRÉE: un ens. S, une fonction $\mathcal{W}$

SORTIE: un booleen

ALGORITHME:

1. $\mathcal{I}' = \mathcal{F}(\mathcal{I})$
2. Appliquer l'algo. $\mathcal{A}$ sur l'instance $\mathcal{I}'$.
3. Si le nombre de sacs est $\geq 3$, retourner faux,
4. sinon, retourner vrai.

Complexité: polynomiale

# Résultat d'inapproximabilité.

**Théorème 3:** Si $P \neq NP$, il n'existe pas d'algorithme polynomiale d'approximation pour le problème Bin Packing ayant un rapport inferieure à $3/2 - \epsilon$.

**Preuve**: par l'absurde.

Soit $\mathcal{A}$ un algo. poly. d'appro. pour Bin Packing ayant un rapport $\alpha \leq 3/2 - \epsilon$.

$\mathcal{A}'$ est un algo. polynomial

Si $A(\mathcal{I}') \leq (3/2 - \epsilon) OPT(\mathcal{I}')$ alors,

- si $OPT(\mathcal{I}') < 3$ alors $A(I) \leq (3/2 - \epsilon)2$
- si $OPT(\mathcal{I}') \geq 3$ alors $3 \leq A(I)$

$\square$

# Résultat d'inapproximabilité.

**Théorème 3:** Si $P \neq NP$, il n'existe pas d'algorithme polynomiale d'approximation pour le problème Bin Packing ayant un rapport inferieure à $3/2 - \epsilon$.

**Preuve**: par l'absurde.

Soit $\mathcal{A}$ un algo. poly. d'appro. pour Bin Packing ayant un rapport $\alpha \leq 3/2 - \epsilon$.

$\mathcal{A}'$ est un algo. polynomial résolvant le prob. partition à partir de $\mathcal{A}$.

Contradiction avec l'hypothèse que $P \neq NP$.

$\square$

# Son appartenance à la classe PTAS?

Une conséquense du théorème pécédent:

**Théorème 3:** Si $P \neq NP$, le problème Bin Packing n'appartient pas à la classe PTAS .

RAPPEL: Soit $\mathcal{P}$ un problème dans NPO. Un algorithme $\mathcal{A}$ est un schéma d'approximation en un temps polynomial pour $\mathcal{P}$ si pour n'importe quelle instance $x$ de $\mathcal{P}$, et pour n'importe rationnel $r > 1$, alors $\mathcal{A}$ ayant comme entré $(x, r)$ retourne une solution ayant un rapport inferieure à $r$. en un temps polynomial en $|x|$.

# Plan

1. Présentation du problème.

2. Algorithme glouton d’approximation.

3. Inapproximation.

4. Remarque sur le codage des données.

# lgo. résouvant le prob. Bin packing ($k = 2$

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \to \mathbb{N}$
un entier $c$

SORTIE: un booléen

ALGORITHME DYNAMIQUE:

Soit $T : n \times c \times c$ un tableau booléen initialisé à $F$.

$T[i, w_1, w_2] = V$ signifie qu'il existe un rangement des $i$ premieres boites telle que

1. les boites du sac $1$ occupe $w_1$ de capacité.
2. les boites du sac $2$ occupe $w_2$ de capacité.

$T[i, w_1, w_2] = V$ ssi $\begin{array}{l} T[i-1, w_1 - w(b_i), w_2] = V \\ \text{ou } T[i-1, w_1, w_2 - w(b_i)] = V \end{array}$

# lgo. résouvant le prob. Bin packing ($k = 2$

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \to \mathbb{N}$
un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau booléen initialisé partout à $F$.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$
3. Pour $i$ allant de $2$ à $n$ faire
   (a) Pour $j, k$ allant de $0$ à $c$ faire
      i. Si $T[i-1, j, k] == V$ alors
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \to \mathbb{N}$
un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau initialisé à $F$.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V;$
3. Pour $i$ allant de $2$ à $n$ faire
   (a) Pour $j, k$ allant de $0$ à $c$ faire
      i. Si $T[i-1, j, k] == V$ alors
      $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
      $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \rightarrow \mathbb{N}$

un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau initialisé à $F$. $0(n \times c^2)$ op.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$; $0(1)$ opérations.
3. Pour $i$ allant de $2$ à $n$ faire
   (a) Pour $j, k$ allant de $0$ à $c$ faire
      i. Si $T[i-1, j, k] == V$ alors
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \rightarrow \mathbb{N}$

un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau initialisé à $F$. $0(n \times c^2)$ op.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$; $0(1)$ opérations.
3. Pour $i$ allant de $2$ à $n$ faire
   (a) Pour $j, k$ allant de $0$ à $c$ faire
      i. Si $T[i-1, j, k] == V$ alors $0(1)$ opérations.
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \to \mathbb{N}$

un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau initialisé à $F$. $0(n \times c^2)$ op.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$; $0(1)$ opérations.
3. Pour $i$ allant de $2$ à $n$ faire
   (a) Pour $j, k$ allant de $0$ à $c$ faire $0(c^2)$ opérations.
      i. Si $T[i-1, j, k] == V$ alors $0(1)$ opérations.
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \to \mathbb{N}$

un entier $c$

SORTIE: un booléen

ALGORITHME:

1. Soit $T : n \times c \times c$ un tableau initialisé à $F$. $0(n \times c^2)$ op.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$; $0(1)$ opérations.
3. Pour $i$ allant de $2$ à $n$ faire $0(n \times c^2)$ opérations.
   (a) Pour $j, k$ allant de $0$ à $c$ faire $0(c^2)$ opérations.
      i. Si $T[i-1, j, k] == V$ alors $0(1)$ opérations.
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux.

# Complexité de l'algorithme.

ENTRÉE: un ens. $\mathcal{B}$ de boites, une fonction $w : \mathcal{B} \rightarrow \mathbb{N}$

un entier $c$

SORTIE: un booléen

ALGORITHME:

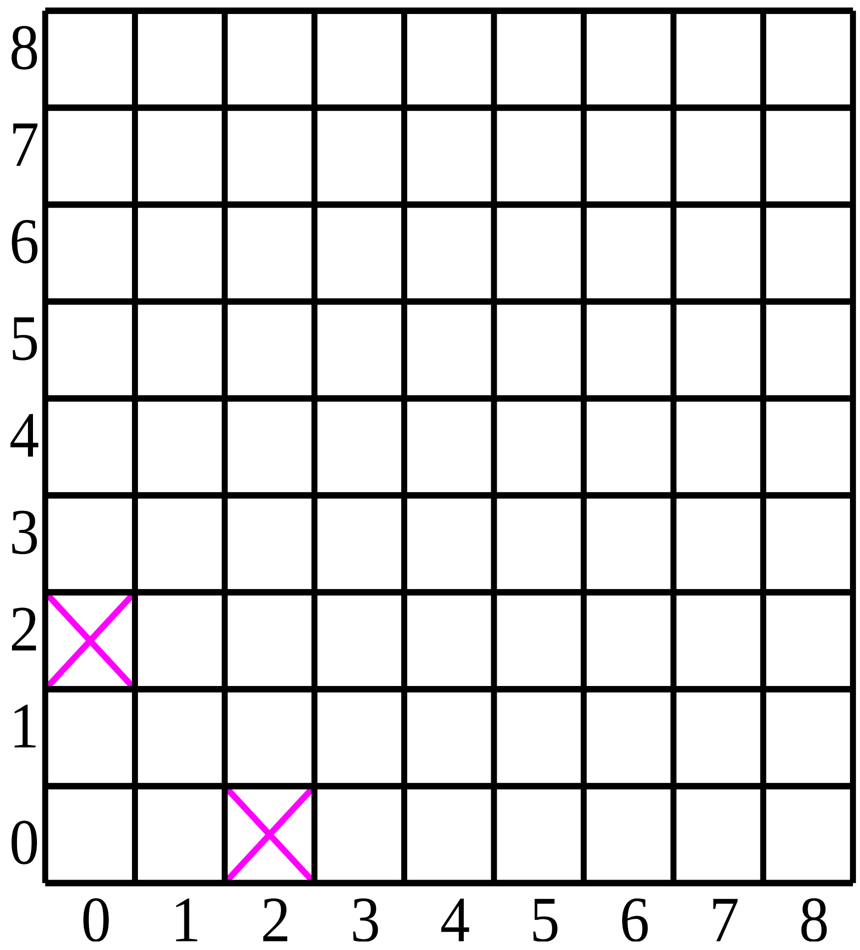
1. Soit $T : n \times c \times c$ un tableau initialisé à $F$. $0(n \times c^2)$ op.
2. $T[1, w(b_1), 0] := T[1, 0, w(b_1)] := V$; $0(1)$ opérations.
3. Pour $i$ allant de $2$ à $n$ faire $0(n \times c^2)$ opérations.
   (a) Pour $j, k$ allant de $0$ à $c$ faire $0(c^2)$ opérations.
      i. Si $T[i-1, j, k] == V$ alors $0(1)$ opérations.
         $T[i, j + w(b_i), k] := V$ si $j + w(b_i) \leq c$
         $T[i, j, k + w(b_i)] := V$ si $k + w(b_i) \leq c$
4. Retourner vrai si $T[n, *, *] = V$ sinon faux. $0(n \times c^2)$ op.

# Illustration de l'algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[1,*,*] =

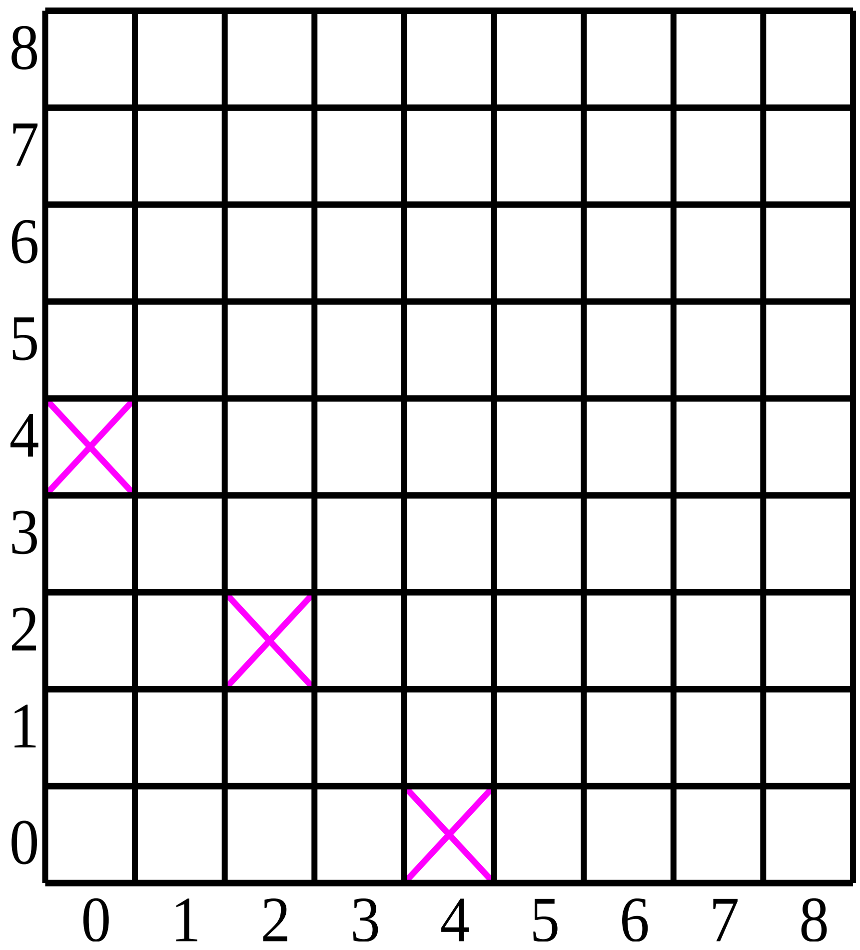


# Illustration de l'algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[2,*,*] =

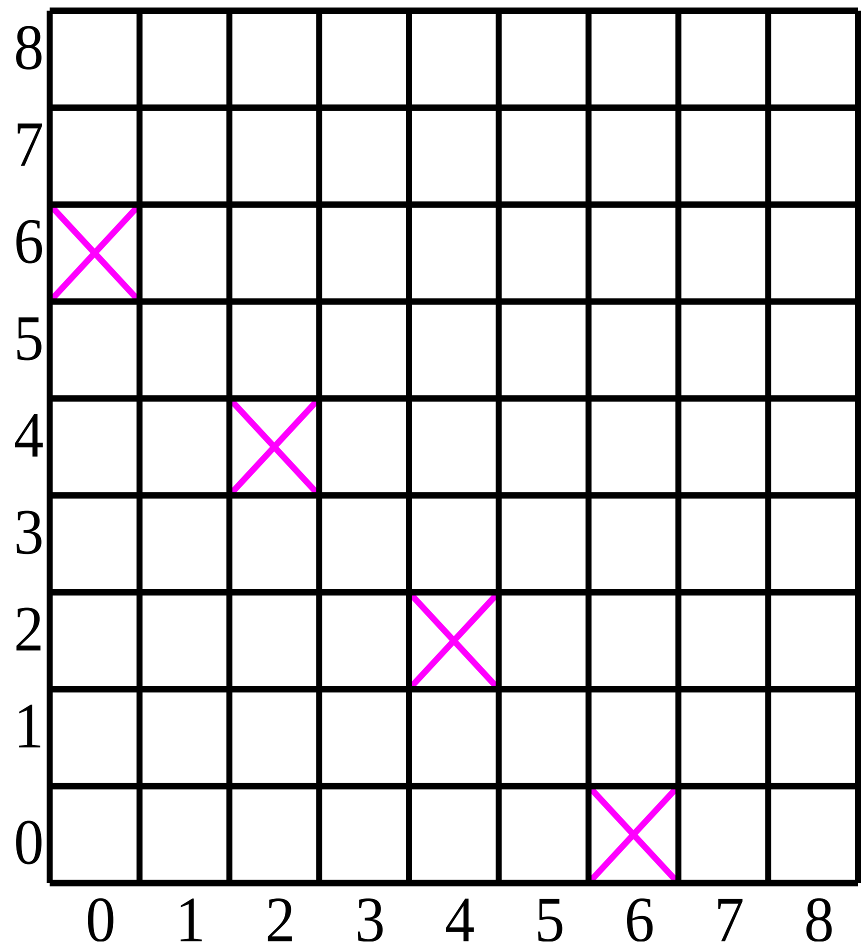


# Illustration de l'algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[3,*,*] =

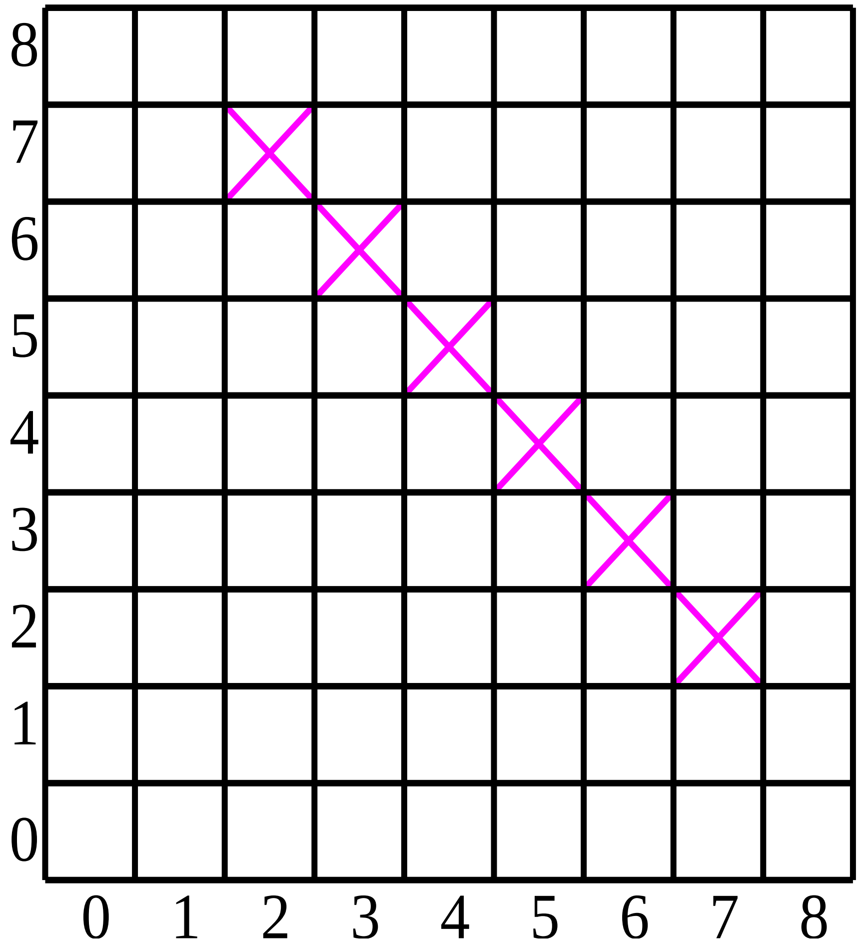


# Illustration de l’algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[4,*,*] =

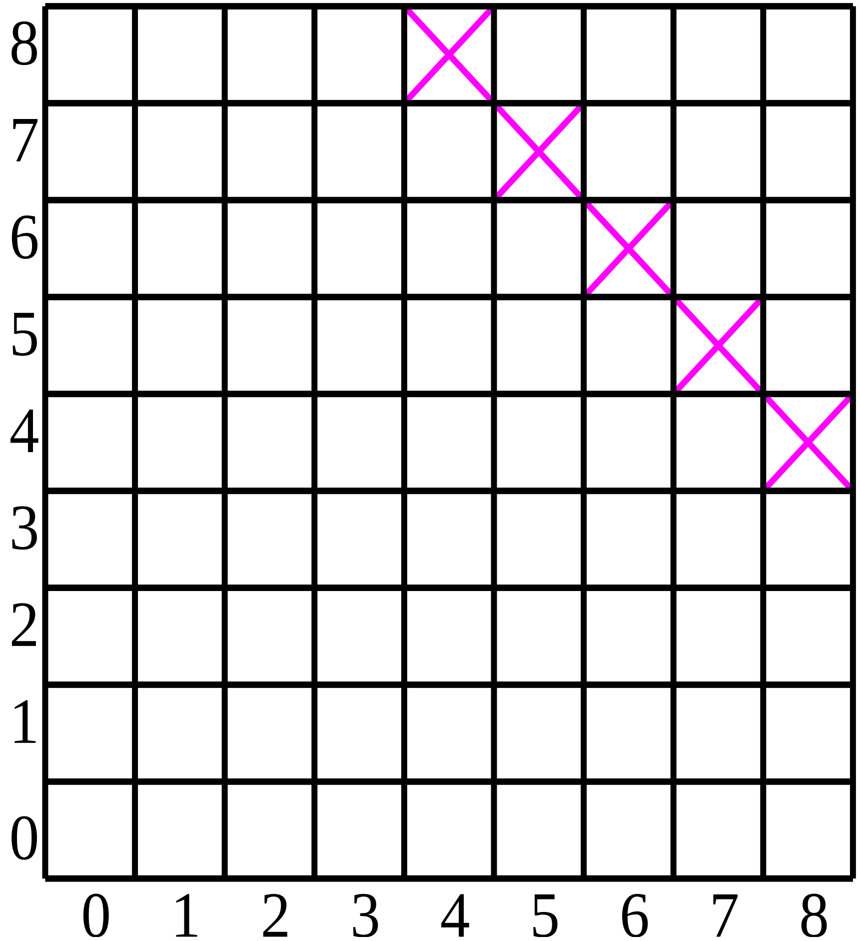


# Illustration de l'algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[5,*,*] =

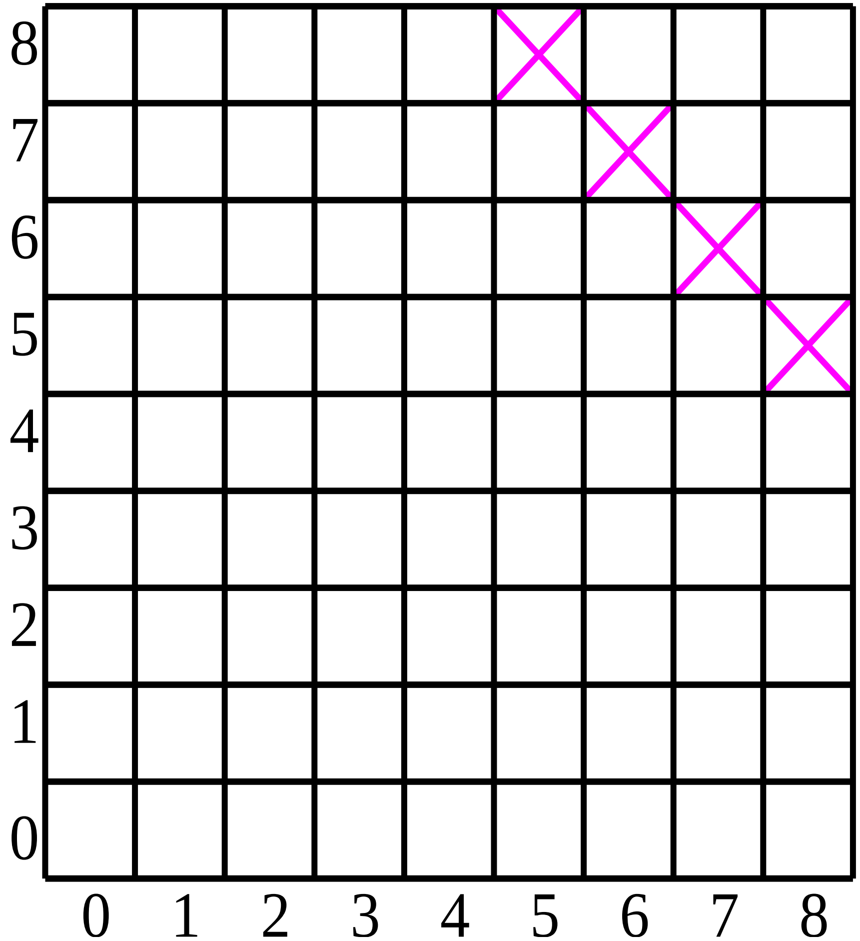


# Illustration de l'algorithme.

$$w(a_1) = w(a_2) = w(a_3) = 2,\ w(a_4) = w(a_5) = 3,\ w(a_6) = 1$$

$$c = 8$$

T[6,*,*] =



# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème de décision Bin Packing:

INSTANCE:

un ensemble $\mathcal{B}$ de boites

une fonction capacité $w : \mathcal{B} \longrightarrow \mathbb{N}$

un entier $c$ et un entier $k = 2$

QUESTION:

Existe-t-il un rangement des boites de $\mathcal{B}$

dans $k$ sacs de capacité $c$?

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème de décision Bin Packing:

INSTANCE:

un ensemble $\mathcal{B}$ de boites

une fonction capacité $w : \mathcal{B} \to \mathbb{N}$

un entier $c$ et un entier $k = 2$

QUESTION:

Existe-t-il un rangement des boites de $\mathcal{B}$

dans $k$ sacs de capacité $c$?

Est-il polynômial?

# Comment coder les entiers ?

**Entiers:** $n$

1. codage en binaire :$|n| = log\ n$.
2. codage exceptionnellement en unaire:$|n| = n$.

Représentation

décimal 13

binaire 1 0 1 1

unaire ● ● ● ● ● ● ● ● ● ● ● ● ●

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème Bin Packing (avec k=2).

EST-IL POLYNÔMIAL?

1. Évaluation de la taille de l'instance:

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème Bin Packing (avec k=2).

EST-IL POLYNÔMIAL?

1. Évaluation de la taille de l'instance:

   INSTANCE:

   un ensemble $\mathcal{B}$ de boites

   une fonction capacité $w : \mathcal{B} \to \mathbb{N}$

   un entier $c$ et un entier $k = 2$

   (a) Si entier codé en binaire : $|I| = O(n \times log_2 c)$

   (b) Si entier codé en unaire : $|I| = O(n \times c)$

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème Bin Packing (avec k=2).

EST-IL POLYNÔMIAL?

1. Évaluation de la taille de l'instance:
   (a) Si entier codé en binaire : $|I| = O(n \times log_2 c)$
   (b) Si entier codé en unaire : $|I| = O(n \times c)$
2. Évaluation de $O(n \times c^2)$ :

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème Bin Packing (avec k=2).

EST-IL POLYNÔMIAL?

1. Évaluation de la taille de l'instance:
   (a) Si entier codé en binaire : $|I| = O(n \times log_2 c)$
   (b) Si entier codé en unaire : $|I| = O(n \times c)$
2. $O(n \times c^2) = \begin{cases} O(c \times |I|) \leq O(|I|^2) & \text{codage unaire} \\ O(c \times e^{|I|}) & \text{codage binaire} \end{cases}$

# En résumé

Il existe un algorithme qui effectue $0(n \times c^2)$ opérations et qui résout le problème Bin Packing (avec k=2).

EST-IL POLYNÔMIAL?

1. Évaluation de la taille de l'instance:
   (a) Si entier codé en binaire : $|I| = O(n \times log_2 c)$
   (b) Si entier codé en unaire : $|I| = O(n \times c)$
2. $O(n \times c^2) = \begin{cases} O(c \times |I|) \leq O(|I|^2) & \text{codage unaire} \\ O(c \times e^{|I|}) & \text{codage binaire} \end{cases}$
3. Il est polynômial si les entiers sont codés en unaire. Mais pas polynômial si le codage binaire des entiers.

# Plan

1. Présentation du problème.
2. Algorithme glouton d'approximation.
3. Inapproximation.
4. Remarque sur le codage des données.
5. Divers.

# Algo. d’approx. sur le bin packing

## SQUELETTE DE L’ALGORITHME

1. Elimininer les boites de petites tailles.
2. Regrouper les boites telles que le nombre de types de boites soit constant
3. Trouver l’optimal pour l’instance restreinte.
4. Dégrouper les boites.
5. Inserer les boites de petites tailles

# Algo. sur un ens restreint d'instances.

**Proposition 1:** Soit un rational $1 \geq \delta > 0$ fixé, soit $K$ un entier positif fixé. Considérons une restriction du problème bin packing

1. dans laquelle un élément est de taille au plus $\delta \times c$ avec $c$ la capacité d'un sac.
2. et dans laquelle le nombre d'éléments distincts est $K$.

Il existe un algorithme polynômial tel que résoud optimalement ce problème restreint.

NOTATION: le problème de bin packing $(K, \delta)$-restreint.

# Preuve de la Proposition 1.

Soit $\mathcal{I}$ une instance du problème $(K, \delta)$-restreint.

1. Une instance peut être défini par un entier $c$ et par un vecteur $\overrightarrow{I} = (s_1 : n_1, s_2 : n_2, \ldots, s_K : n_K)$
   - $n_i$: nombre de boites de type $s_i$.
   - $c$ est la capacité d'un sac.
2. Exemple d'une instance $(3, 3/8)$-restreinte.

   Une instance définit par $\overrightarrow{I} = (3 : 4, 5 : 2, 7 : 1)$ et $c = 8$
   - Cette instance contient $3$ types d'éléments
     - $4$ éléments de taille $3$
     - $2$ éléments de taille $5$
     - $1$ élément de taille $7$

# Preuve de la Proposition 1.

Soit $\mathcal{I}$ une instance du problème $(K, \delta)$-restreint.

1. Une instance peut être défini par un entier $c$ et par un vecteur $\overrightarrow{I} = (s_1 : n_1, s_2 : n_2, \ldots, s_K : n_K)$
   - $n_i$: nombre de boites de type $s_i$.
   - $c$ est la capacité d'un sac.
2. Un type de sac peut être défini par un vecteur d'entiers $\overrightarrow{t} = (t_1, t_2, \ldots, t_K)$ tel que
   - Ce sac contient $t_i$ boites de type $s_i$.
   - $\sum_{i=1}^{K} t_i \times s_i \leq c$
   - $\forall i : 1 \leq i \leq K, \delta \times c \leq s_i$

# Preuve: Type de sac

1. Un type de sac peut être défini par un vecteur d'entiers $\overrightarrow{t} = (t_1, t_2, \ldots, t_K)$ tel que
   - Ce sac contient $t_i$ boites de type $s_i$.
   - $\sum_{i=1}^{K} t_i \times s_i \leq c$
   - $\forall i : 1 \leq i \leq K, \delta \times c \leq s_i$
2. remarque pour chaque $\overrightarrow{t}$,

$$\forall i : 1 \leq i \leq K, 1 \leq \frac{s_i}{\delta \times c} \Rightarrow \sum_{i=1}^{K} t_i \leq \frac{1}{\delta} \sum_{i=1}^{K} t_i \frac{s_i}{c} \leq \frac{1}{\delta}$$

$$\sum_{i=1}^{K} t_i \leq \frac{1}{\delta}$$

# Preuve de la Proposition 1.

1. Remarque: pour chaque $\overrightarrow{t}$: $\sum_{i=1}^{K} t_i \leq \frac{1}{\delta}$.
2. Donc le nombre $q$ de manière pour choisir $K$ entiers tel que la somme est inferieure ou égal à $\lfloor 1/\delta \rfloor$ est:

$$q = \binom{K + \lfloor 1/\delta \rfloor}{\lfloor 1/\delta \rfloor}$$

3. Inst. $(3, 3/8)$-restreinte: $\overrightarrow{I} = (3:4, 5:2, 7:1)$ et $c = 8$
   - $K = 3$, $\lfloor 1/\delta \rfloor = 2$, donc $q = \binom{5}{2} = 10$
   - En fait, il existe $6$ types de sacs possibles: $(0,0,0)$,$(0,0,1)$,$(0,1,0)$,$(1,0,0)$,$(1,1,0)$,$(2,0,0)$

# Preuve de la Proposition 1.

1. $q$ (nb de types de sacs) dépends de $K$ et de $\delta$
2. Une solution acceptable $\overrightarrow{y} = (y_1, \ldots, y_q)$ où
   - $y_i$ représente un sac du type $i$.
   - $0 \leq y_i \leq n$ avec $n$ nombre d'éléments ( boîtes).
3. Nombre de solutions acceptables borné par $O(n^q)$ avec $n =$ nbre de boîtes
4. Algorithme$=$ Enumérer toutes les solutions possibles et selectionner l'optimale.

Compléxité: $O(n^q p(n))$ avec $p$ un polynome en $n$

# Algo. d'approximation.

1. Soit $I$ une telle instance telle que $\mathcal{B} = \{b_1, \ldots, b_n\}$ est trié dans l'ordre décroissant en fct de $w$.
2. $\forall \alpha \leq n$, $m = \left\lfloor \frac{n}{\alpha} \right\rfloor$, partionner les $n$ boîtes en $m+1$ groupes $G_i$ avec $G_i = \left\{b_{(i-1)\alpha+1}, \ldots, b_{i\alpha}\right\}$ et $G_{m+1} = \{b_{m\alpha+1}, \ldots, b_n\}$.
3. Instance $J$ = Instance $I$ modifiée:

   tous les éléments d'un même groupe $i$, $2 \leq i \leq m+1$ a la capacité la plus grande de ce groupe dans $I$.
4. $J$ possède $m$ types différents de boîtes.
5. Application de l'algorithme optimal sur $J$.

# Exemple: grouper

Soit $x = ((9, 9, 8, 7, 6, 6, 5, 4, 3, 3, 3), c)$ et $\alpha = 3$

Transformer l'instance $x$ en une instance restreinte $x_g$:

instance de départ $x = ((9, 9, 8, 7, 6, 6, 5, 4, 3, 3, 3), c)$

$G_1 = \{9, 9, 8\} \quad G_2 = \{7, 6, 6\} \quad G_3 = \{5, 4, 3\} \quad G_4 = \{3, 3\}$

lissage

$G_2 = \{7, 7, 7\} \quad G_3 = \{5, 5, 5\} \quad G_4 = \{3, 3\}$

instance restreinte: $x_g = ((7, 7, 7, 5, 5, 5, 3, 3), c)$

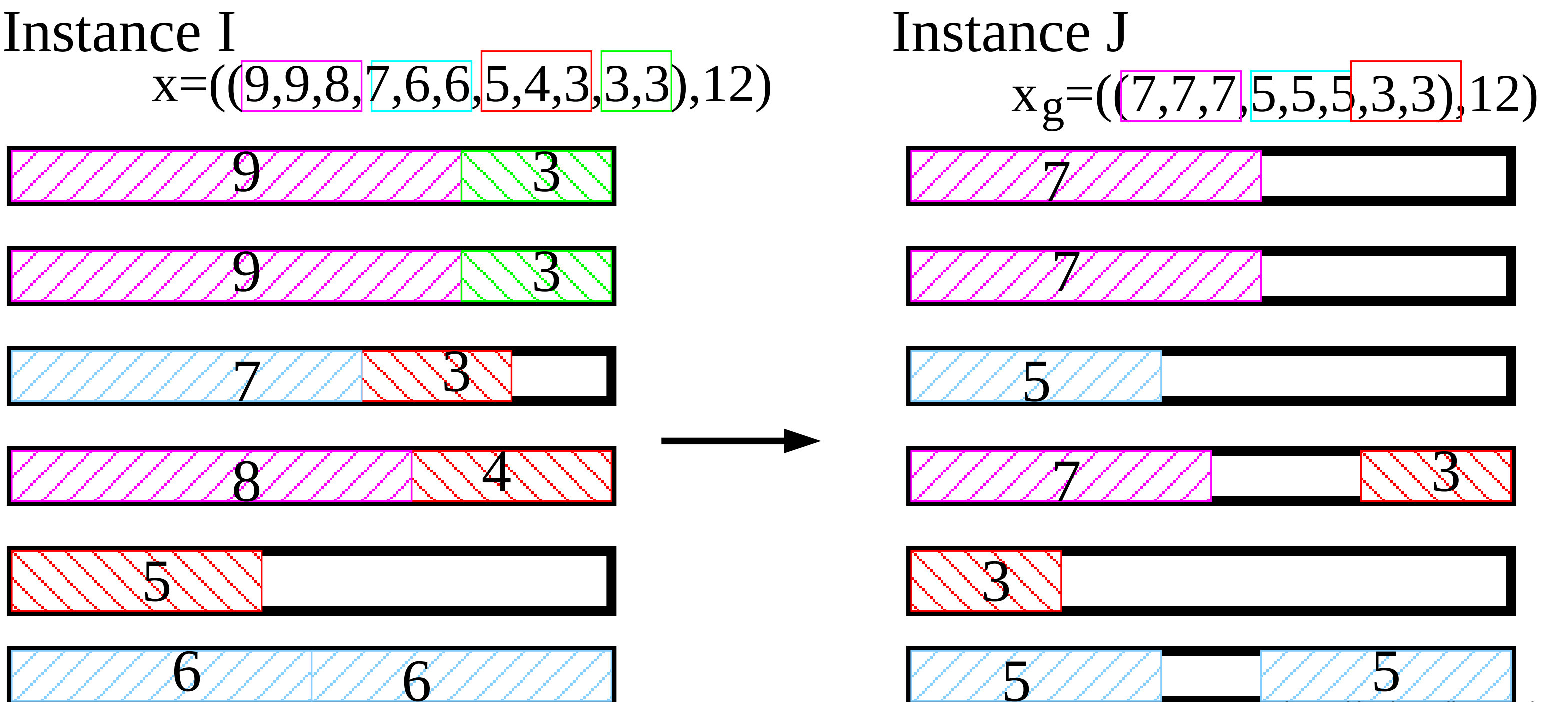
# Comparaison entre l’optimal de $I$ et de $J$.

1. Chaque rangement de $I$ peut être remplacé par un rangement de $J$:

# Comparaison entre l'optimal de $I$ et de $J$.

1. Chaque rangement de $I$ peut être remplacé par un rangement de $J$:

   si $b \in G_i$ est rangé dans le sac $j$ pour l'instance $I$, alors, il existe une boite $b' \in G_{i+1}$ est rangé dans le sac $j$ pour l'instance $J$



# Comparaison entre l'optimal de $I$ et de $J$.

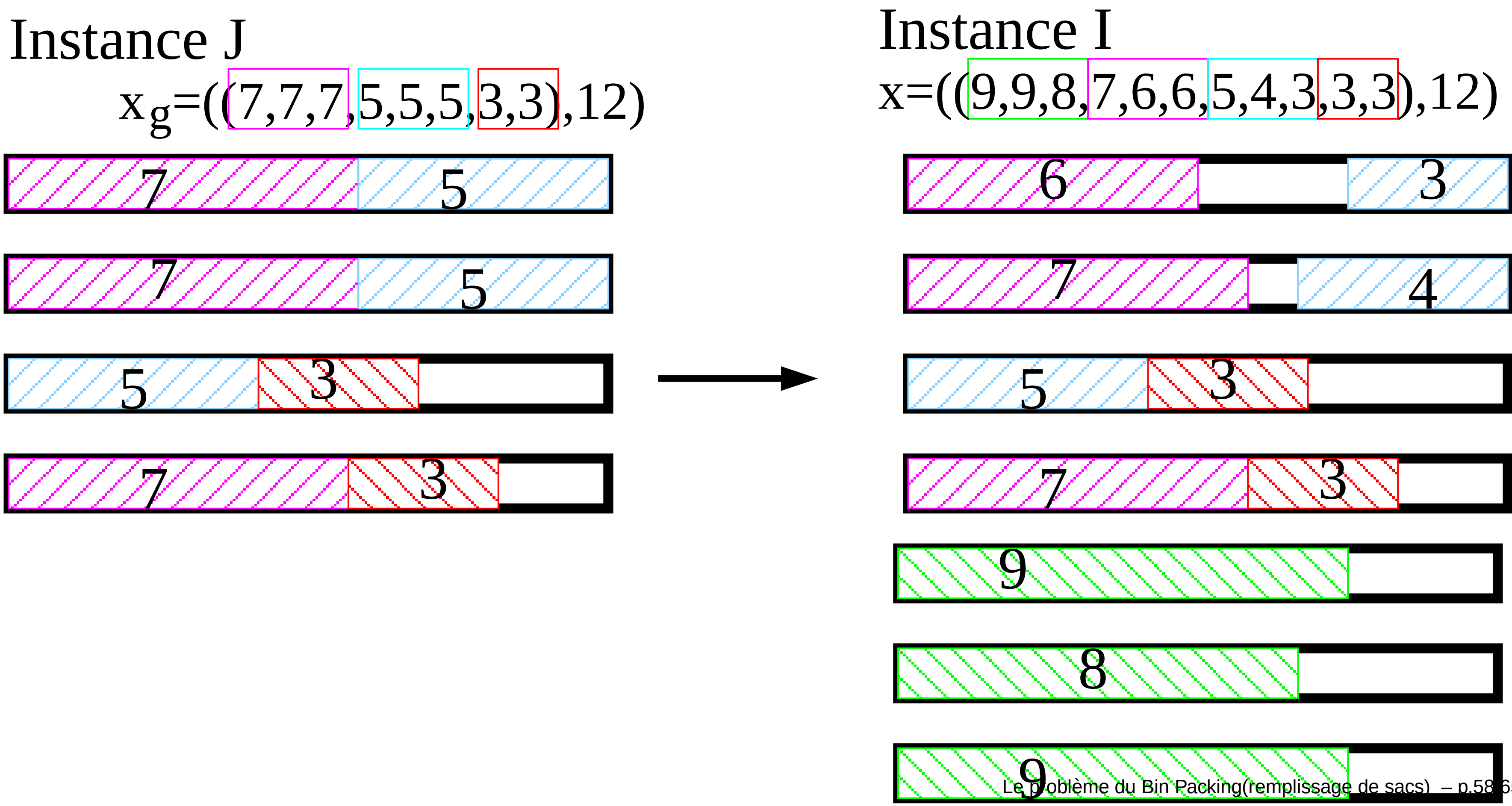
1. Chaque rangement de $I$ peut être remplacé par un rangement de $J$:

$$m^*(J) \leq m^*(I)$$

2. Chaque rangement de $J$ peut être transformé en un rangement de $I$ plus $\alpha$ sacs:

# Comparaison entre l'optimal de $I$ et de $J$.

2. Chaque rangement de $J$ peut être transformé en un rangement de $I$ plus $\alpha$ sacs:



# Comparaison entre l'optimal de $I$ et de $J$.

Donc:

$$m^*(J) \leq m^*(I) \leq m^*(J) + \alpha$$

# Traitement des petites boîtes.

1. Soit $I$ une instance du bin packing, et $\delta \in (0, 1/2]$
2. $I_\delta$= Instance obtenue par élimination des boites de capacité $< \delta \times c$.
3. Soit $\mathcal{R}$ un rangement optimal de $I_\delta$ utilisant $M$ sacs.
4. Utilisation l'approche FIRST FIT pour construire un rangement $\mathcal{R}'$ pour $I$.

# Traitement des petites boîtes.

1. Soit $I$ une instance du bin packing, et $\delta \in (0, 1/2]$
2. $I_\delta$= Instance obtenue par élimination des boites de capacité $< \delta \times c$.
3. Soit $\mathcal{R}$ un rangement optimal de $I_\delta$ utilisant $M$ sacs.
4. Utilisation l'approche FIRST FIT pour construire un rangement $\mathcal{R}'$ pour $I$.

QUESTION: Comment évaluation du nombre de sacs crées par l'algorithme FIRST FIT?

# 2 cas possibles.

1. Aucun sac a été créé par l'algorithme FIRST FIT: $\mathcal{R}'$ utilise $M$ sacs.
2. Sinon Soit $M'$ le nbre de sacs créé par cette procédure:
   (a) Tous les sacs de $M' \cup M$ excepté $1$ a une place vide au plus $\delta c$.
   (b) Donc

$$(1-\delta)(M+M'-1) \leq \frac{\sum_{i=1}^{n} w(b_i)}{c} \leq m^*(I)$$

$$(M+M') \leq \frac{1}{1-\delta} m^*(I) + 1 \leq (1+2\delta)m^*(I) + 1$$

# 2 cas possibles.

1. Aucun sac a été créé par l'algorithme FIRST FIT: $\mathcal{R}'$ utilise $M$ sacs.
2. Sinon Soit $M'$ le nbre de sacs créé par cette procédure:
   (a) Tous les sacs de $M' \cup M$ excepté $1$ a une place vide au plus $\delta c$.
   (b) Donc $(M + M') \leq (1 + 2\delta)m^*(I) + 1$
   (c) En conclusion, la solution est calculée en tps polynomial tel que le nombre de sacs est majoré par

$$max(M, (1 + 2\delta)m^*(I) + 1)$$

# Algorithme

ENTRÉE: 1 instance $I$: $n$ boites, entier $c$, rational $1 < r < 2$.

SORTIE: une solution tel que la mesure $rOPT + 1$

ALGORITHME:

1. $\delta \leftarrow (r-1)/2$
2. $J \leftarrow I \backslash \{$de boîtes de cap. $< \delta c\}$
3. $\alpha \leftarrow \left\lceil \frac{(r-1)^2 n'}{2} \right\rceil$ avec $n'$ de nbre de boîtes de $J$.
4. $J'$: instance obtenue par groupement par $\alpha$ boîtes de $J$
5. Trouver la solution optimale pour $J'$
6. Inserer les $\alpha$ première boites dans $\alpha$ nouveaux sacs
7. Appliquer Algo First Fit pour inserer les petites boites.
8. retourner le rangement obtenu.

# Conclusion

**Théorème 4:** L'algorithme précedent est un schéma d'approximation en temps polynomiale asymptotiquement .

PREUVE

# Conclusion

**Théorème 4:** L'algorithme précedent est un schéma d'approximation en temps polynomiale asymptotiquement .

PREUVE

- Algorithme polynomiale

# Conclusion

**Théorème 4:** L'algorithme précedent est un schéma d'approximation en temps polynomiale asymptotiquement .

PREUVE

- Algorithme polynomiale :
  - Construire les nouvellles instances $J$ et $J'$: $O(n)$.
  - calculer l'optimal de $J'$: $O(n^q p(n))$
    - $q$ dépend de $r$
    - $p$ est un polynome.

# Conclusion

**Théorème 4:** L'algorithme précedent est un schéma d'approximation en temps polynomiale asymptotiquement .

PREUVE

- Algorithme polynomiale
- Qualité de la solution notée $m(I)$:

# Conclusion: Evaluation de la qualité.

1. au départ, instance $I$ et $\delta = (r-1)/2$
2. $J \leftarrow$ Elimininer les boites de petites tailles ($< \delta c$) .
3. $J' \leftarrow$ Regrouper les boites en $\alpha = \left\lceil \frac{(r-1)^2 n'}{2} \right\rceil$ boites + lissage.
4. Trouver l'optimal pour l'instance restreinte $J'$.
5. Adapter la sol. pour $J$.
6. Inserer les boites de petites tailles

# Conclusion: Evaluation de la qualité.

1. au départ, instance $I$ et $\delta = (r-1)/2$
2. $J \leftarrow$ Elimininer les boites de petites tailles ($< \delta c$) .
   $m^*(J) \leq m^*(I)$
3. $J' \leftarrow$ Regrouper les boites en $\alpha = \left\lceil \frac{(r-1)^2 n'}{2} \right\rceil$ boites + lissage.
4. Trouver l'optimal pour l'instance restreinte $J'$. $m^*(J')$
5. Adapter la sol. pour $J$.$m(J) = m^*(J') + \alpha$
6. Inserer les boites de petites tailles

# Conclusion: Evaluation de la qualité.

1. au départ, instance $I$ et $\delta = (r-1)/2$
2. $J \leftarrow$ Elimininer les boites de petites tailles ($< \delta c$) .
   $m^*(J) \leq m^*(I)$
3. $J' \leftarrow$ Regrouper les boites en $\alpha = \left\lceil \frac{(r-1)^2 n'}{2} \right\rceil$ boites + lissage.
4. Trouver l'optimal pour l'instance restreinte $J'$. $m^*(J')$
5. Adapter la sol. pour $J$.$m(J) = m^*(J') + \alpha$
6. Inserer les boites de petites tailles
   $m(I) = max(m(J), (1+2\delta)m^*(I) + 1)$

# Conclusion: Evaluation de la qualité.

1. $m(J) = m^*(J') + \alpha$ avec $\alpha = \left\lceil \frac{(r-1)^2 n'}{2} \right\rceil$
2. tous les éléments de $J$ ont une capacité $> \delta c$ avec $\delta = (r-1)/2$.

   Donc $\delta n' \leq m^*(J)$ car $\frac{\delta \times c \times n'}{c} \leq \frac{\sum_{b \in J} w(b)}{c} \leq m^*(J)$
3. $\alpha \leq \frac{(r-1)^2 n'}{2} + 1 \leq (r-1) \times \delta n' + 1 \leq (r-1) m^*(J) + 1$
4. $\alpha \leq (r-1) m^*(J) + 1$
5. Donc $m(J) \leq r \times m^*(J) + 1$

# Conclusion: Evaluation de la qualité.

**Théorème 4:** L'algorithme précedent est un schéma d'approximation en temps polynomiale asymptotiquement .

PREUVE: Qualité de la solution notée $m(I)$?

1. $m(I) = max(m(J), (1+2\delta)m^*(I)+1)$ avec $\delta = (r-1)/2$.
2. $m(J) \leq r \times m^*(J) + 1$
3. $m(I) \leq max(r \times m^*(J) + 1, (1+2\delta)m^*(I)+1)$
4. $m(I) \leq max(r \times m^*(J) + 1, rm^*(I)+1)$ avec $r = 2\delta + 1$
5. $m(I) \leq rm^*(I) + 1$ avec $m^*(J) \leq m^*(I)$