Algorithmes d'approximation

Motivation

- On connaît un certain nombre de problèmes **de décision** NP-complets:
  - $3-SAT, CLIQUE, VERTEX-COVER$
  - $COLOR, CHEMIN-HAMILTONIEN$
- Si $P \neq NP$, il n'y a pas d'algorithme polynomial pour aucun d'entre eux.

- La plupart d'entre eux sont associés à des problèmes d'optimisation.
- Exemple: pour VERTEX-COVER, le problème **d'optimisation** associé est:
  - Instance: Un graphe
  - Question: Un sous-ensemble de sommets *de taille minimale* telle que toute arête soit adjacente à l'un d'entre eux.
- Théorème: Si $P \neq NP$, alors le problème **d'optimisation** ne possède pas d'algorithme polynomial.

Preuve

- En effet, étant donné G, k, si il y avait un tel algorithme on pourrait calculer la taille k_0 du sous-ensemble retourné par l'algorithme sur G, et répondre vrai si et seulement si $k_0 \leq k$, pour résoudre le problème **de décision VERTEX-COVER**.
- Puisque $VERTEX - COVER$ est NP-complet, cela impliquerait $P = NP$, absurde.

Algorithme d'approximation

- Un algorithme de ρ-approximation est un algorithme polynomial qui renvoie une solution approchée garantie au pire cas à un facteur ρ de l'optimal.
- Autrement dit, si on note Opt la solution optimale, et $Algo$ la solution retournée par l'algorithme, on a

$$Opt \leq Algo \leq \rho Opt.$$

Un algorithme GLOUTON-COVER

- Algorithme GLOUTON-COVER(G), avec le graphe $G = (V, E)$.
  - $S := \emptyset$
  - tant qu'il y a des arêtes non-couvertes
    - prendre une arête $e = (x, y)$ non-couverte
    - faire $S := S \cup \{x, y\}$
    - détruire les arêtes adjacentes aux sommets x et y dans G.
  - Retourner S.
- Théorème: $GLOUTON - COVER$ est une 2-Approximation du problème d'optimisation associé à VERTEX-COVER.

Preuve

- Soit A l'ensemble des arêtes choisies par $GLOUTON - COVER$.
- On a $|S \models 2 * |A|$, car chaque arête ajoute deux sommets, et les arêtes de A n'ont aucun sommet en commun.
- Or $Opt \geq |A|$ car il faut au moins couvrir les arêtes de A.
- Donc $|S| \leq 2Opt$.

Problème du voyageur de commerce

Un représentant doit visiter n villes. Le représentant souhaite faire une tournée en visitant chaque ville au moins et exactement 1 fois et en terminant à sa ville de départ.

Traduction en un problème de décisio

- Problème du Voyageur de Commerce (TSP=Traveling Salesman Problem)
  - Instance:

    $G = (V, E)$ un graphe complet

    une fonction coût $c : V \times V \rightarrow \mathbb{N}$

    un entier $k \in \mathbb{N}$
  - Question:

    Existe-t-il un tournée $\mathcal{H}$ de coût $\leq k$?
- Théorème: Le problème du Voyageur de Commerce est NP-complet.

Pb d'optimisation associé

- Instance:

  $G = (V, E)$ un graphe complet

  une fonction coût $c : V \times V \rightarrow \mathbb{N}$

- Question:

  Une tournée $\mathcal{H}$ de côut minimal.

- Théorème: Si $P \neq NP$, ce problème ne possède pas d'algorithme polynomial.
- (preuve: même principe que pour $VERTEX - COUVER$).

Inapproximabilité

Théorème: Si $P \neq NP$ et $\rho \geq 1$, il n'existe aucun algorithme de ρ-approximation pour le problème d'optimisation associé au problème du voyageur de commerce.

Preuve

Par l' absurde. Supposons qu'il existe un algo. d'approx. B de borne ρ.
Construisons une instance du voyageur de commerce à partir d'une instance du problème du cycle hamiltonien.
$\mathcal{F}(G = (V, E)) =$

$$\begin{cases} G' = (V, E') \; graphe \; complet \\ c(u,v) = \begin{cases} 1 & si \; (u,v) \in E \\ 1 + \rho|V| & sinon \end{cases} \end{cases}$$

Suite (1/3)

1. Si G a un cycle hamiltonien, (G', c) a une tournée de coût $|V|$
2. Sinon, le coût d'une tournée $\geq (1 + \rho|V|) + (|V| - 1) > \rho|V|$

Suite (2/3)

Algo. polynomial déterminant si G a un cycle hamiltonien,

1. $(G', c) := \mathcal{F}(G)$.
2. Soit H la tournée calculée par B ayant en entrée (G', c)
3. retourner $\begin{cases} vrai & si\ c(H) \leq \rho|V| \\ faux & sinon\ (c(H) > \rho|V|) \end{cases}$

- solutions possibles

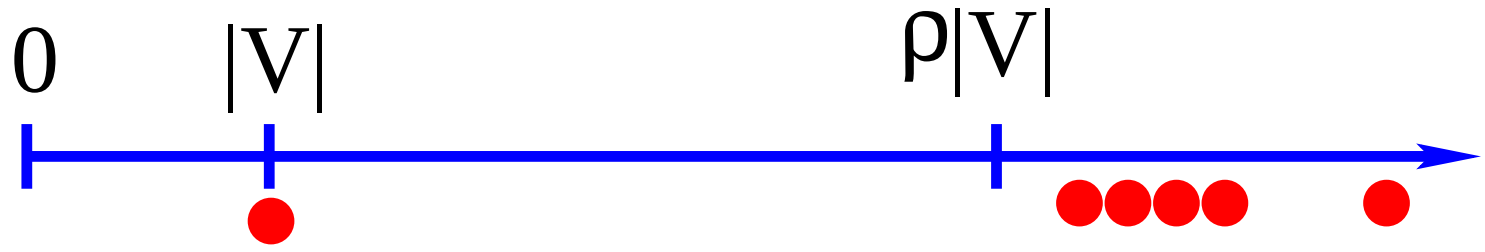
Suite (3/3): Remarque

1. Si $c(H) \leq \rho|V|$, cela signifie que $c(H^*) \leq |V|$,

   où H^* la tournée optimale est un cycle hamiltonien de G.

2. Si $c(H) > \rho|V|$, cela signifie que $c(H^*) > |V|$,

   G ne peut pas possèder de cycle hamiltonien.



● solutions possibles

On obtient une contradiction: l'algorithme précédent est un algorithme polynomial pour le problème $CYCLE - HAMILTONIEN$ NP-complet, contredisant l'hypothèse $P \neq NP$.

Cas Particulier

Définition:
La fonction de coût vérifie l'inégalité triangulaire si

pour 3 sommets u, v, w arbitraires de V,

$$c(u, v) \leq c(u, w) + c(w, v).$$

Théorème: Le problème du Voyageur de Commerce avec la fonction de coût vérifiant l'inégalité triangulaire est NP-complet.

Algorithme

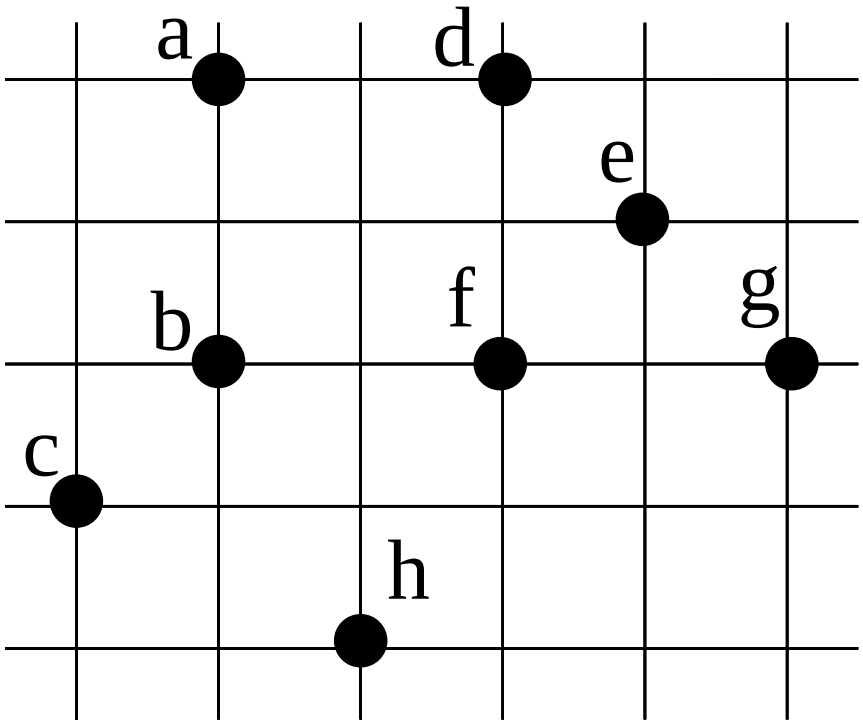
- **Entrée:** $G = (V, E)$ un graphe complet
  une fonction coût $c : V \times V \to \mathbb{N}$
  qui vérifie l'inégalité triangulaire

Sortie: une tournée.

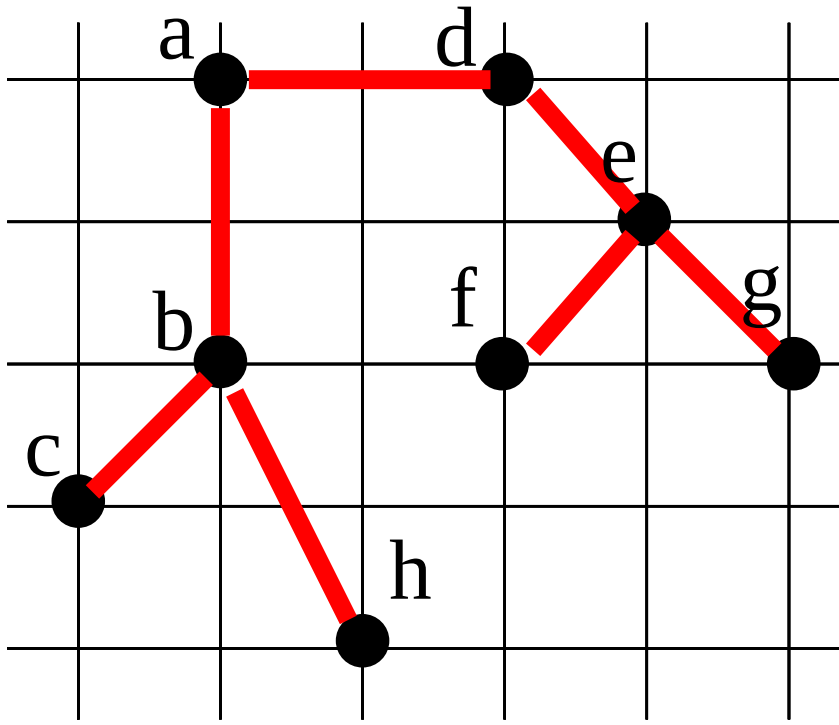
Algorithme:

1. Choisir un sommet $r \in V$ comme racine.
2. Construire un arbre couvrant minimal T dans G à partir de la racine r.
3. Soit L la liste des sommets visités lors d'un parcours préfixe.
4. Retourner le cycle hamiltonien H qui visite les sommets dans l' ordre de L.

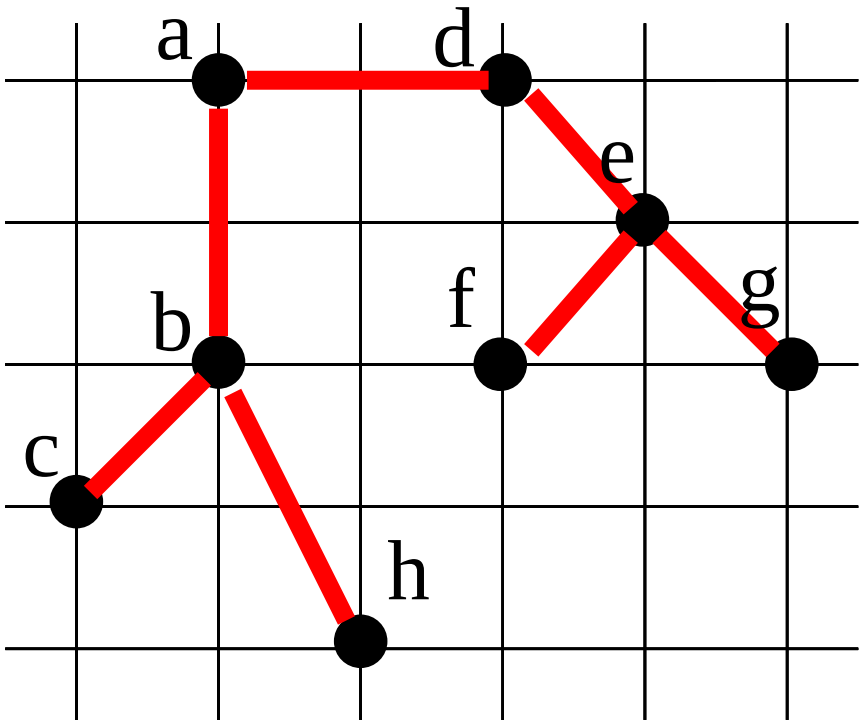
Illustration de l'algorithme



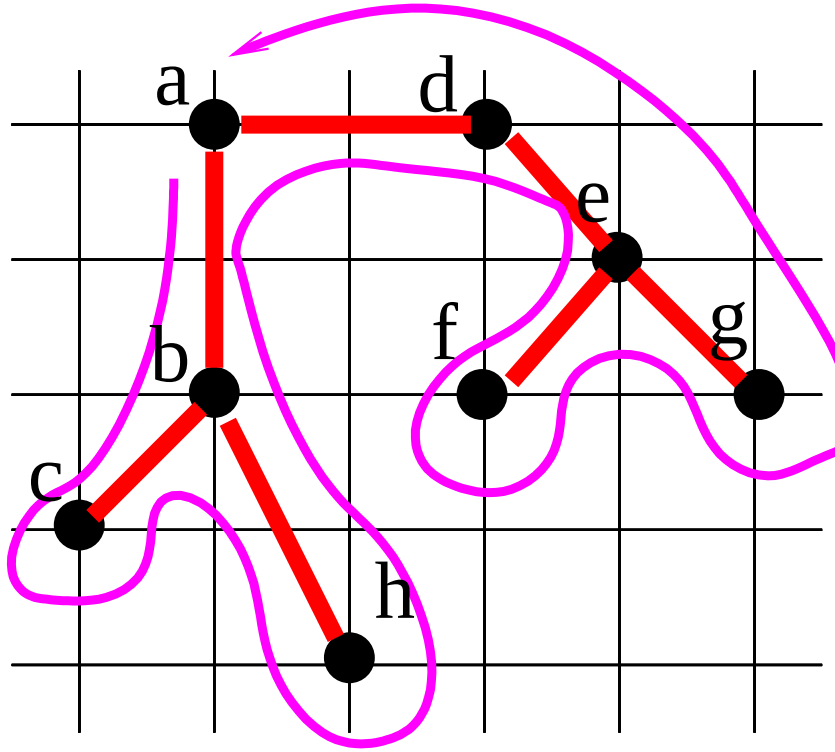
instance



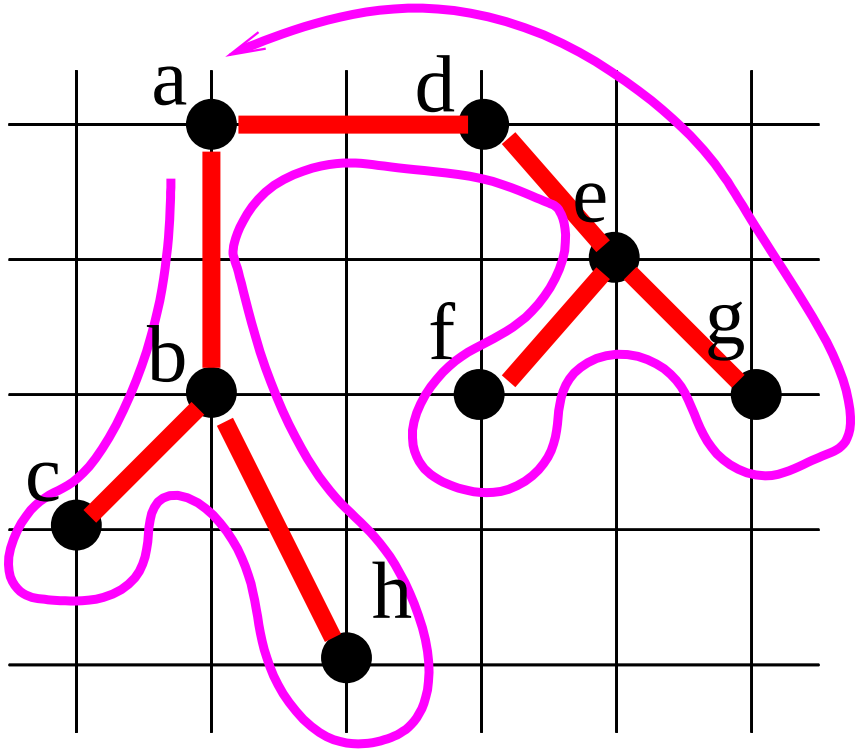
arbre couvrant



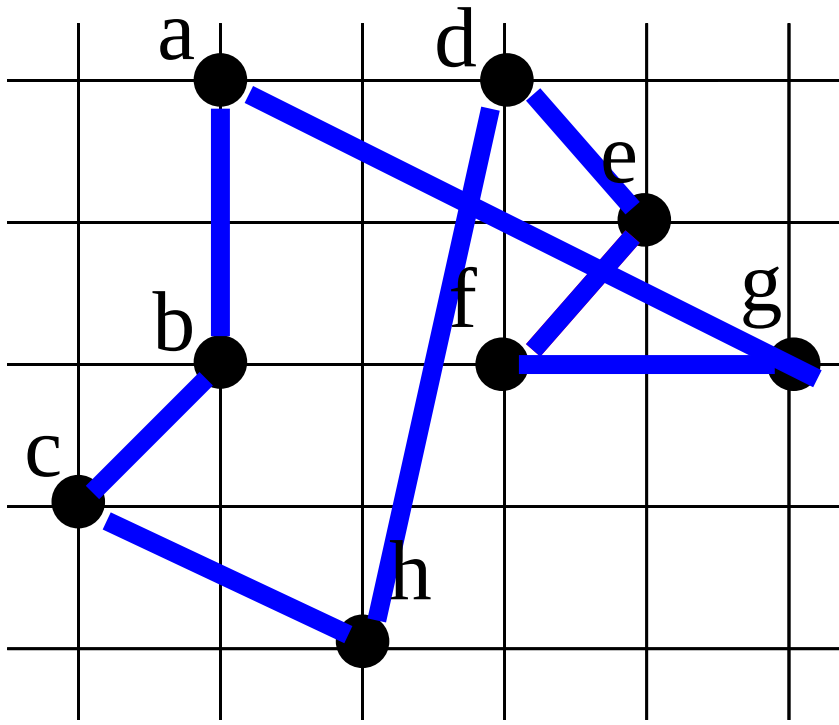
arbre couvrant



parcours préfixe



parcours préfixe



solution

Facteur d'approximation

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

Proof:

Soit H^* la tournée optimale.

Il reste donc à prouver que $c(H) \leq 2c(H^*)$.

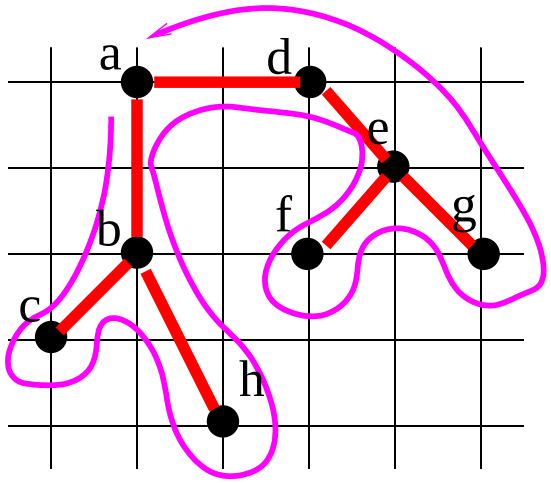
Soit T un arbre couvrant minimal: $c(T) \leq c(H^*)$ (car le cycle H^* privé d'une arête est un arbre).

Soit L le parcours de l'arbre T (ajout d'un sommet dans la liste dès que l'on les rencontre lors du parcours).

□

Facteur d'approximation (suite)

parcours $L = a, b, c, b, h, b, a, d, e, f, e, g, e, d, a.$



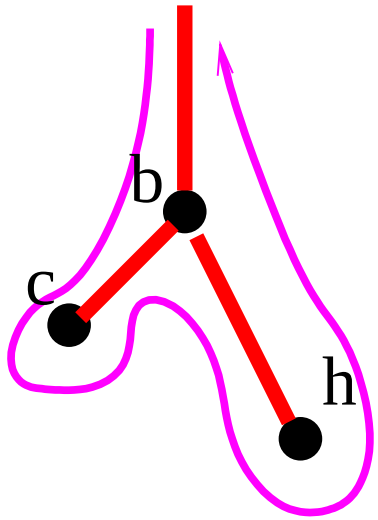
$c(L) = 2c(T)$ car chaque arête est visitée 2 fois.

$c(H) \leq 2c(H^*)$?

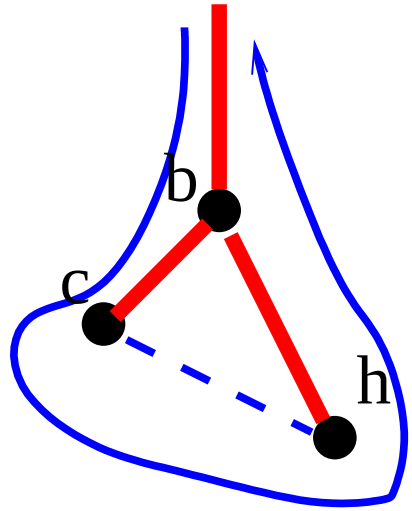
- Donc $c(L) \leq 2c(H^*)$.
- On peut supprimer un sommet de L sans augmenter le coût (grâce à l'inégalité triangulaire).
- Donc $c(H) \leq c(L) \leq 2c(H^*)$.

Plus précisement,

On peut supprimer 1 sommet de W sans augmenter le coût (grâce à l'inégalité triangulaire).



....b c b h b....



....b c h b....

$$c(b,c) + c(c,h) + c(h,b) + \ldots \leq$$
$$c(b,c) + c(c,b) + c(b,h) + c(h,b) + \ldots$$

car $c(c,h) \leq c(c,b) + c(b,h)$ grâce à l'inégalité triangulaire

Énoncé du problème Bin Packing

Problème: mettre un ensemble de boites de tailles données dans des sacs de même capacité, en utilisant le moins possible de sacs.

Énoncé du problème Bin Packing

Instance:

un ensemble $\mathcal{B}$ de boites

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

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 $\mathcal{B} \rightarrow [1...k]$ tel que

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

Complexité

- Théorème: Le problème du bin Packing est NP-complet.
- Preuve:
  1. Problème est dans NP: exercice.
  2. Réduction avec le problème Partition, NP-complet.
     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 S_1 et S_2 tels que $\sum_{s \in S_1} \mathcal{W}(s) = \sum_{s \in S_2} \mathcal{W}(s)$?

$PARTITION \leq BIN - PACKING$ via la transformation polynômiale $\mathcal{F}$

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

On a

$(S, \mathcal{W}) \in PARTITION$ ssi $\mathcal{F}(S, \mathcal{W}) \in BIN - PACKING$.

Illustration de la transformation

Partition

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

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

$\rightarrow$

Bin Packing

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

$w_{\ (a_1, \ldots, a_5)} = \{6, 7, 2, 4, 1\}$

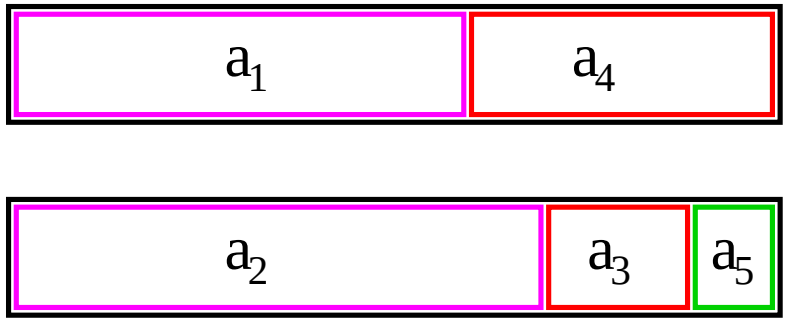
$c = 10,\ k = 2$

Réponse: oui car

$S_1 = \{a_1, a_4\}$

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

Réponse: oui car



Résultats

- Le problème $BIN - PACKING$ est NP-complet.
- Il est 2-approximable.
- Pour tout $1 < \lambda < 3/2$, il n'est pas λ-approximable.

L'algorithme *Next Fit.*

Entrées: 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:

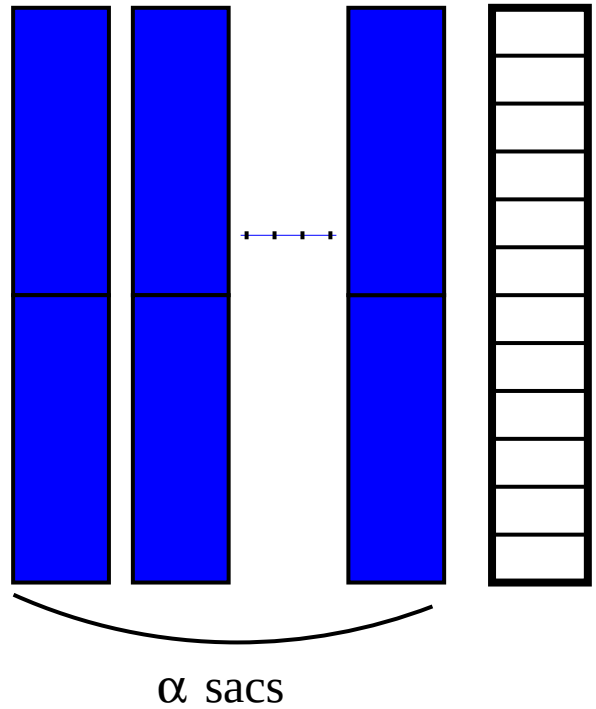
deux entiers α, et n tel que $n = 4\alpha$,

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

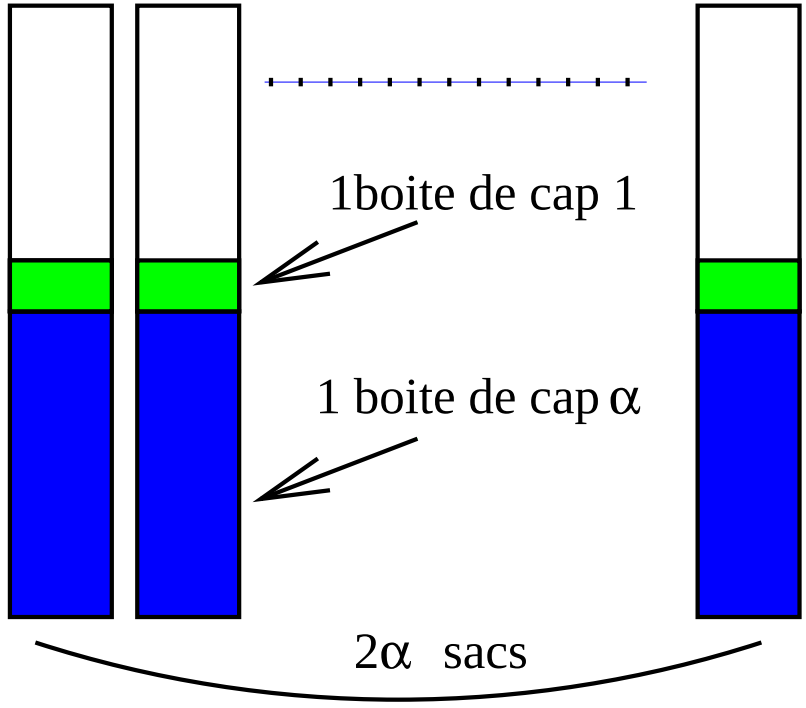
une fonction w tel que $w(\mathcal{B}) = \{\alpha, 1, \alpha, 1, \ldots, \alpha, 1\}$,

$c = 2\alpha$

Solution optimale: $\alpha + 1$



Solution donnée par $NextFit$: 2α



Facteur d'approx. de $NextFit$

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

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

- Si on somme sur j, $kc < \sum_{j=1}^{k} \sum_{\ell=\alpha}^{i+1} w(b_\ell)$
- Donc $kc < 2 \sum_{b \in \mathcal{B}} w(b)$ (car chaque $w(b)$ apparaît au plus deux fois dans la somme).
- D'où $k \leq 2\beta/c \leq 2OPT$.

L'algorithme *First Fit Decreasing*.

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

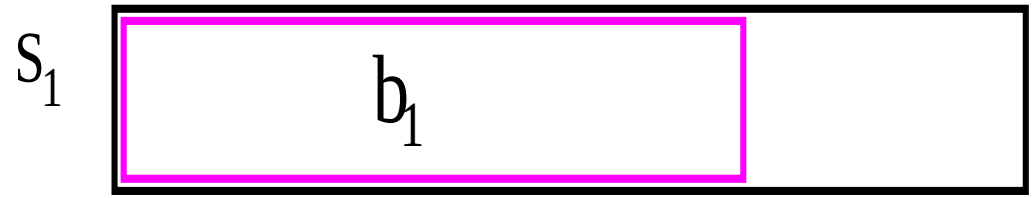
Sortie: un entier k

Algorithme:

1. Trier $\mathcal{B} = \{b_1, \ldots, b_n\}$ dans l'ordre décroissant suivant 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 $S_{k+1} = \{b_i\}$, $k := k + 1$.

Illustration de l'algorithme

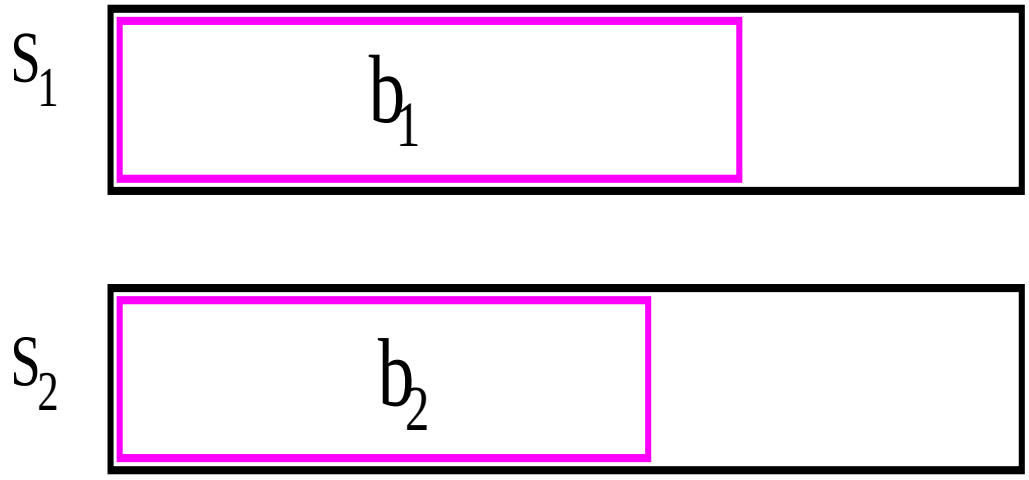
Exemple: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



k=1

Illustration de l'algorithme

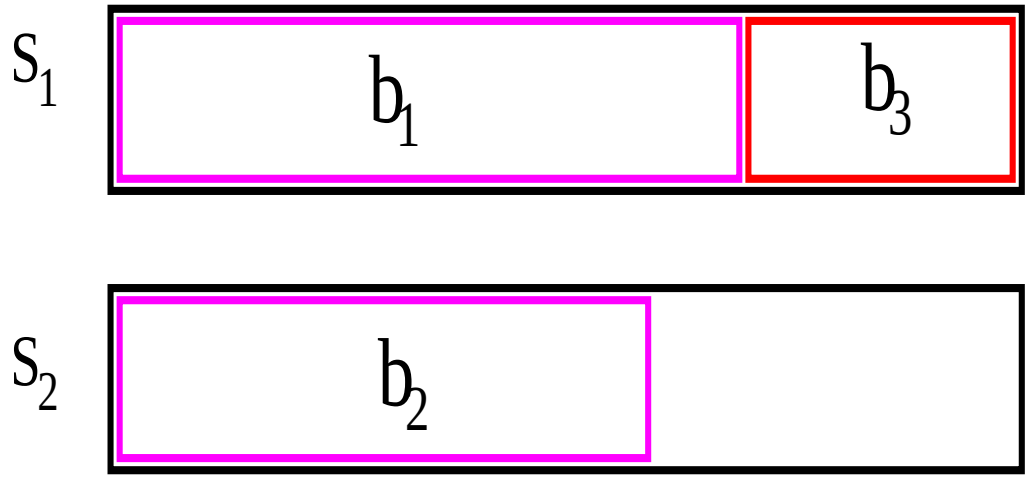
Exemple: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



k=1

Illustration de l'algorithme

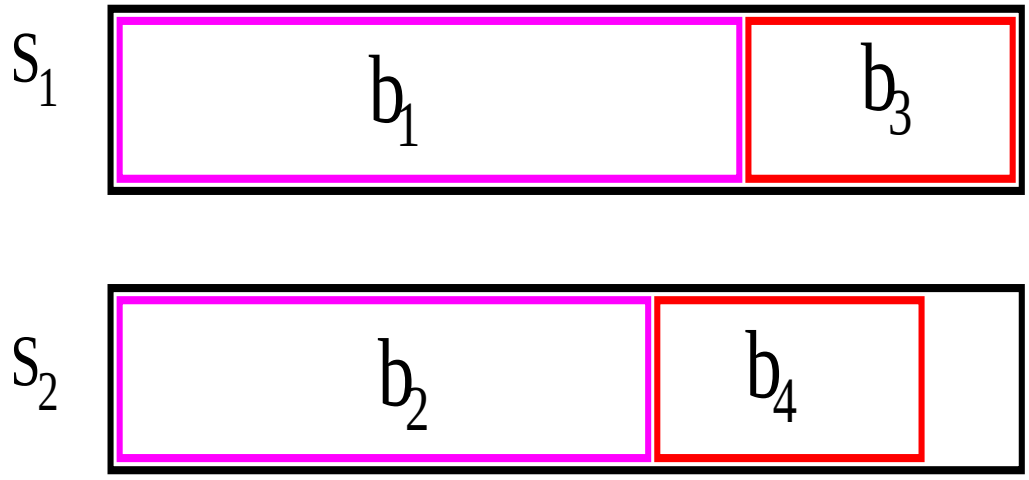
Exemple: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



k=2

Illustration de l'algorithme

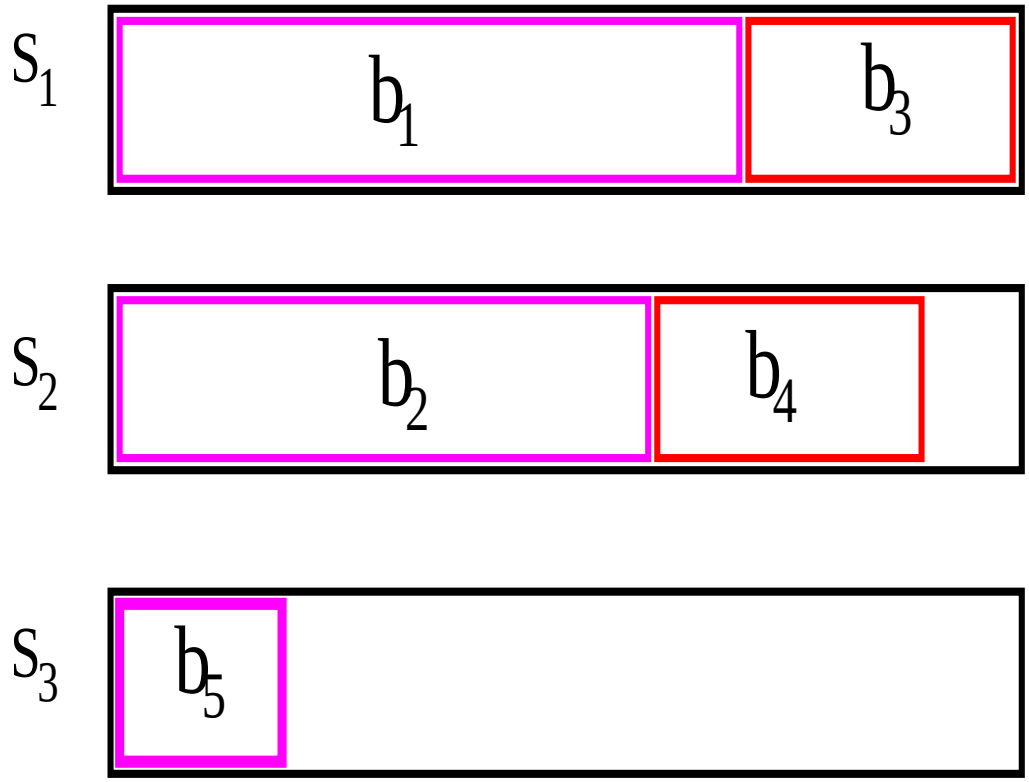
Exemple: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



k=2

Illustration de l'algorithme

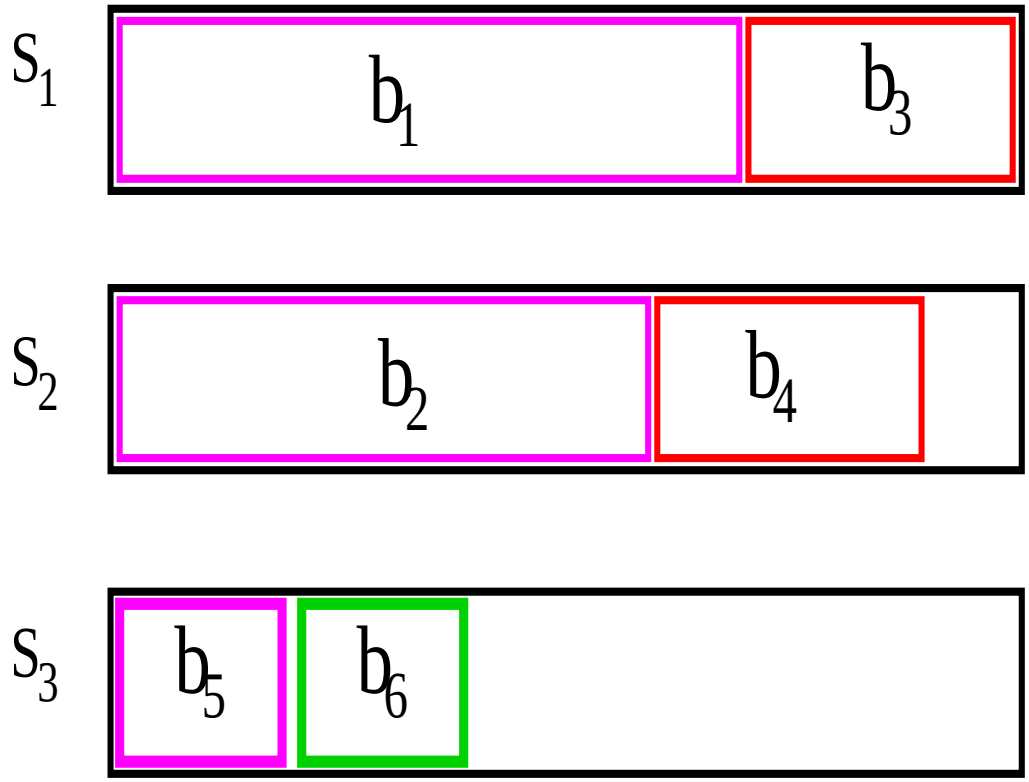
Exemple: $c = 10, w(b_1) = 7$, $w(b_2) = 6$, $w(b_3) = w(b_4) = 3$, $w(b_5) = w(b_6) = 2$.



k=3

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

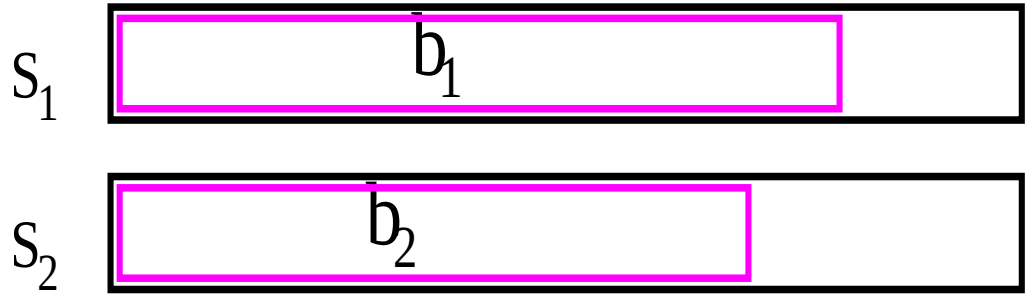
Remarque: $OPT \geq (\sum_{b \in \mathcal{B}} w(b))/c$.
Car la borne inférieure correspond à une solution où tous les sacs sont pleins.

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

1 élément par sac.

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

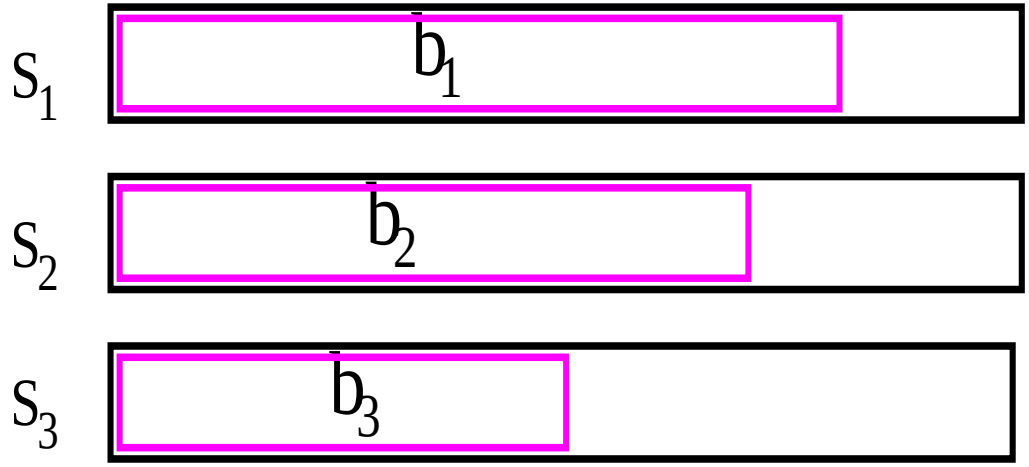
Cas 1 où $w(b_n) \geq c/2$: l'algorithme glouton est optimal.



1 élément par sac.

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

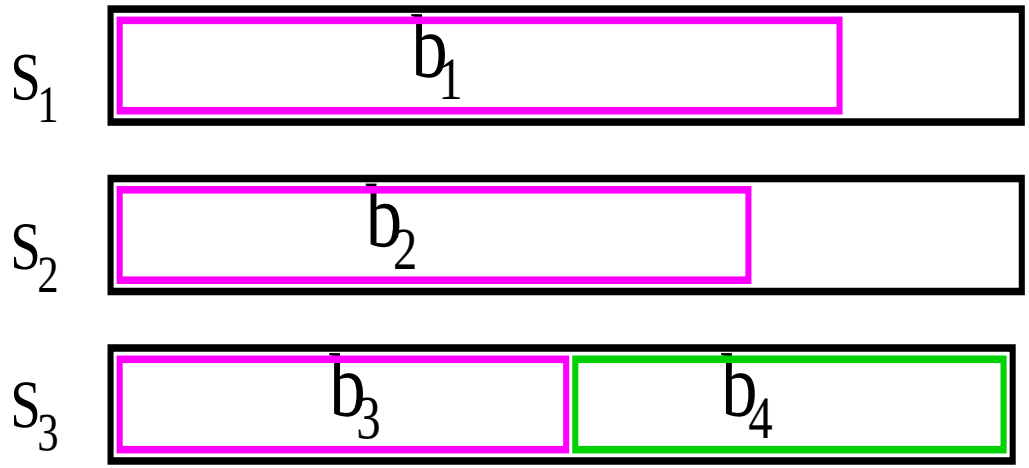
Cas 1 où $w(b_n) \geq c/2$: l'algorithme glouton est optimal.



Au plus 2 éléments par sac.

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

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



Au plus 2 éléments par sac.

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

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

Instance:

deux entiers α, et n tel que $n = 5\alpha$,
un ens. $\mathcal{B} = \{b_1, \ldots, b_n\}$ de boites,
une fonction w telle que

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

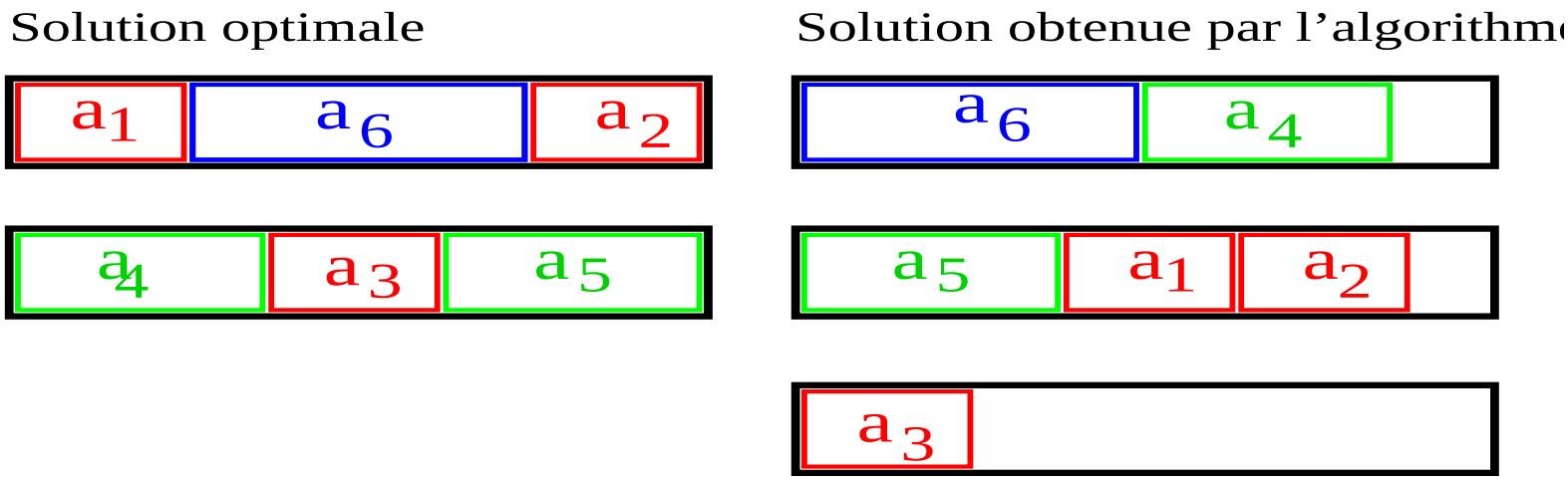
$c = 2\alpha$

Solution optimale: $= \frac{3}{2}\alpha$

Solution fourni par l'algo: $= \frac{11}{6}\alpha$

Plusieurs questions se posent alors...

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



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

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

Transformation

Objectif: Transformer 1 instance de *Partition* en 1 instance du *Bin Packing*.

Transformation $\mathcal{F}(S, \mathcal{W}) =$

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

Proposition: 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,

≥ 3 sinon.

Preuve

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

Résultat d'inapproximabilité.

- Théorème: 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 résout le problème partition à partir de $\mathcal{A}$.

Algorithme $\mathcal{A}'$.

Entrée: un ensemble S, une fonction $\mathcal{W}$

Sortie: un booléen

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 ≥ 3, retourner faux,
4. sinon, retourner vrai.

Complexité: polynomiale

- 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 $3 \leq A(I)$.

- L'algorithme $\mathcal{A}'$ est donc un algorithme polynomial correct pour $PARTITION$ NP-complet: contradiction avec l'hypothèse que $P \neq NP$.