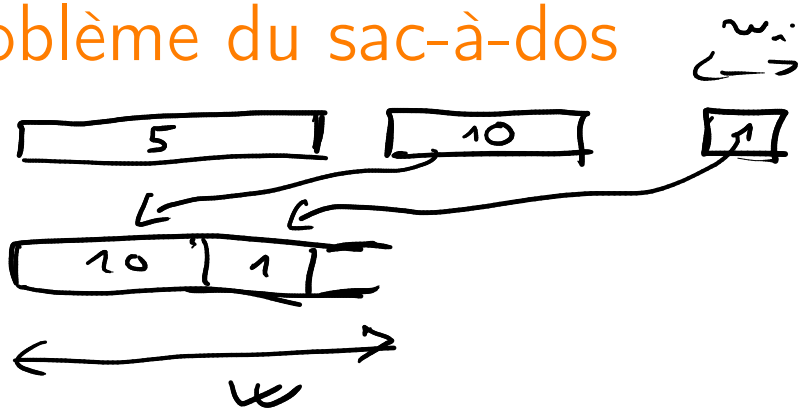
8. Sac-à-dos, bin-packing, et entrepôts

13 Décembre 2023

1. Sac à dos
2. Bin Packing
3. Positionnement d'entrepôts
4. Exercices

Problème du sac-à-dos



Problème du sac-à-dos

- **Instance**. Entiers positifs ou nuls n, $w_1, \ldots, w_n$ et W, et des réels positifs ou nuls $c_1, \ldots, c_n$.
- **Question**. Trouver un sous-ensemble $S \subseteq \{1, \ldots, n\}$ tel que $\sum_{j \in S} w_j \leq W$ et $\sum_{j \in S} c_j$ est maximum.

Applications (et généralisations en plusieurs dimensions)

1. Gestion de portefeuilles / investissements
2. Génération de clefs en cryptographie
3. Choix de questions à traiter dans un exam noté sur plus de 20.
4. Technologie embarquée.
5. Etc.

Complexité

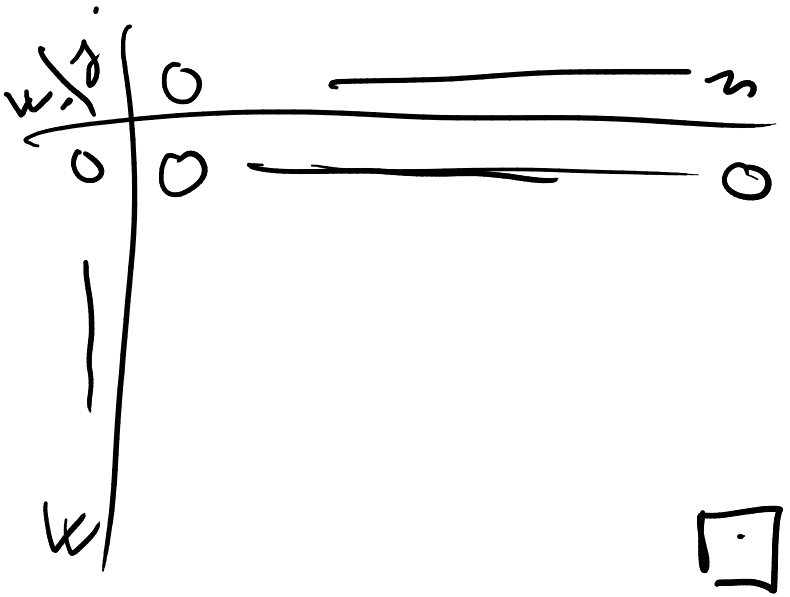
Théorème. Le problème du sac-à-dos est NP-difficile.

Programmation dynamique

Quelle fonction valeur pourrait être utilisée pour une approche par programmation dynamique ?

$V(j, w')$ = valeur optimale du problème restreint aux j premiers objets et avec un sac de taille w'

$$\begin{cases} V(0, w') = 0 \\ V(j, w') = \end{cases}$$



- Si j est dans la solution optimale pour (j, w')

$$V(j, w') = V(j-1, w'-w_j) + c_j$$

- Si j n'est pas dans la

$$V(j, w') = V(j-1, w')$$

Donc $V(j, w') = \max\left(V(j-1, w'), V(j-1, w'-w_j) + c_j\right)$

$O(n\,W)$ ← ⟶

$$|I|_2 = \sum \lceil \log_2 w_i \rceil + \lceil \log_2 c_i \rceil + \lceil \log_2 W \rceil$$

$$\geq \log_2 W$$

$$2^{|I|_2} \geq W$$

Programmation dynamique

$V(j, W')$ = valeur maximale d'un sac de capacité W', en se limitant aux j premiers objets.

Equation (programmation dynamique)

$$V(j, W') = \max(V(j-1, W'), V(j-1, W' - w_j) + c_j).$$

Complexité : $O(nW)$.

Question

Pourquoi l'algorithme n'est il pas polynomial ?

Programmation dynamique

$V(j, W')$ = valeur maximale d'un sac de capacité W', en se limitant aux j premiers objets.

Equation (programmation dynamique)

$$V(j, W') = \max(V(j-1, W'), V(j-1, W' - w_j) + c_j).$$

Complexité : $O(nW)$.

Question

Pourquoi l'algorithme n'est il pas polynomial ?

Ce n'est pas polynomial car W nécessite $\log_2(W)$ bits pour être codés. C'est un algorithme **pseudo-polynomial**.

Formulation PLNE

Donner un programme linéaire en nombres entiers pour ce problème.

$$\begin{cases} \max_{x} \sum_{i=1}^{n} c_i x_i \\ \text{s.c.} \sum_{i=1}^{n} w_i x_i \leq W \\ x_i \in \{0,1\} \quad \forall i \in [\![1, n]\!] \end{cases}$$

Formulation PLNE

Donner un programme linéaire en nombres entiers pour ce problème.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.c.} \quad & \sum_{j=1}^{n} w_j x_j \leq W \\ & x_j \in \{0,1\} \qquad \text{pour tout } j \in \{1,\ldots,n\} \end{aligned}$$

Calcul de la relaxation linéaire

La borne supérieure

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.c.} \quad & \sum_{j=1}^{n} w_j x_j \leq W \\ & 0 \leq x_j \leq 1 \qquad \text{pour tout } j \in \{1, \ldots, n\} \end{aligned}$$

peut facilement se calculer avec un solveur.

Exercice

Donner un algorithme très simple qui calcule la solution du relâché continu.

$$\rightarrow \quad \frac{c_1}{w_1} \geq \dots \geq \frac{c_j}{w_j} \geq \dots \geq \frac{c_n}{w_j}$$

$$\underline{j} = \underset{j}{\operatorname{argmax}} \sum_{i=1}^{j} w_j \leq W$$

Solution optimale du relâché :

$$\forall i \in [\![1, j]\!], \quad x_i = 1$$

$$x_{j+1} = \frac{W - \sum_{i=1}^{j} w_j}{w_{j+1}}$$

Solution du relâché continu

- Classer les objets de façon à ce que

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \ldots \geq \frac{c_n}{w_n}. \qquad O(n \log n)$$

- Poser $x_1 = x_2 = \cdots = x_{\bar{j}} := 1$ avec $\bar{j}$ le plus grand entier tel que $\sum_{j=1}^{\bar{j}} w_j \leq W$.
- Poser $x_{\bar{j}+1} := \frac{W - \sum_{j=1}^{\bar{j}} w_j}{w_{\bar{j}+1}}$.
- Poser $x_{\bar{j}+2} = \ldots = x_n := 0$.

Proposition. Un tel x est solution optimale de la relaxation.

Heuristique avec garantie

Algorithme fournissant une solution valant toujours au moins la moitié de la valeur optimale :

On suppose qu $w_j \leq W$ pour tout j.

On reprend l'indexation par $\frac{c_j}{w_j}$ et le $\bar{j}$ du relâché linéaire.

- $\sum_{j=1}^{\bar{j}+1} c_j$ est une borne supérieure de la valeur optimale OPT.
- $\{1, \ldots, \bar{j}\}$ et $\{\bar{j}+1\}$ sont deux solutions réalisables, l'une des deux étant par conséquent au moins de valeur $\geq \frac{1}{2} \sum_{j=1}^{\bar{j}+1} c_j$, i.e. de valeur $\geq \frac{1}{2} OPT$.

$$\sum_{i=1}^{\bar{j}} c_j + c_{\bar{j}+1} > OPT$$

Donc $\sum_{i=1}^{\bar{j}} c_j > \frac{1}{2} OPT$ ou $c_{\bar{j}+1} > \frac{1}{2} OPT$

On a une solution t_q $\frac{1}{2} OPT < SOL \leq OPT$

1 Sac à dos

2 Bin Packing

3 Positionnement d'entrepôts

4 Exercices

Formalisation du bin-packing

BIN-PACKING

- **Instance**. Entiers positifs ou nuls $a_1, \ldots, a_n$ et W.
- **Question**. Trouver un entier naturel k et une fonction $f : [n] \to [k]$ avec $\sum_{i:\, f(i)=j} a_i \leq W$ pour tout $j \in [k]$ tels que k soit minimum.

Interprétation : conteneurs de taille W, objets de tailles a_i. Mettre tous les objets dans un nombre minimum de conteneurs (f indique l'affectation des objets aux conteneurs).

Applications (et généralisation en plusieurs dimensions)

Application 1. Stockage optimal (entrepôts), transport optimal (camions) ; souvent, il y a des contraintes de forme ou de volume en plus (bin-packing 2D ou 3D)

Application 2. Découpe : on a des poutres de longueur fixée, on a une commande pour des longueurs $a_1, \ldots, a_n$; minimiser le nombre de poutres utilisées.

Application 3. Copier un disque dur avec le moins de clés USB possible.

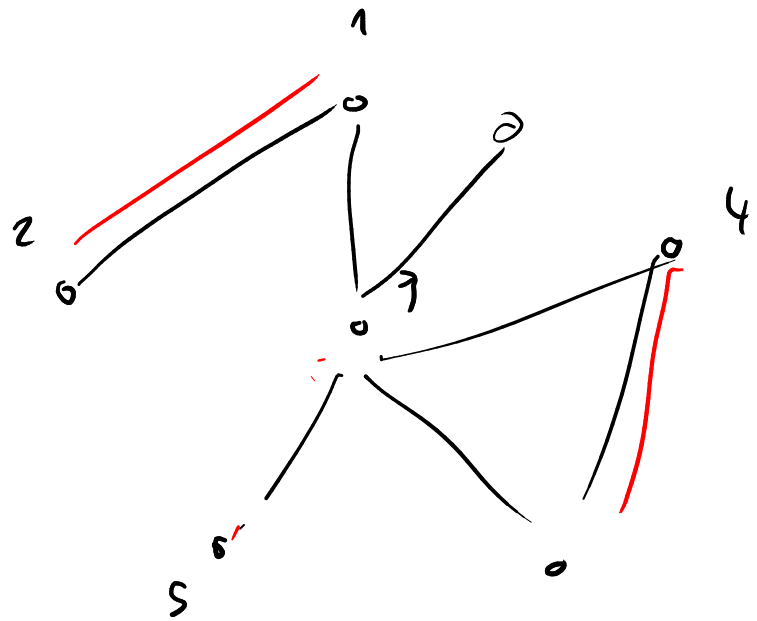
Complexité

Théorème. BIN-PACKING est NP-difficile.

Un cas particulier facile

Exercice

Supposons qu'un instance $a_1, \ldots, a_n$ du bin-packing soit telle que $\frac{a_i}{W} > \frac{1}{3}$ pour tout $i = 1, \ldots, n$. Donner un algorithme polynomial pour résoudre ce cas particulier.


1
2
3
4
5

Un cas particulier facile

Exercice

Supposons qu'un instance $a_1, \ldots, a_n$ du bin-packing soit telle que $\frac{a_i}{W} > \frac{1}{3}$ pour tout $i = 1, \ldots, n$. Donner un algorithme polynomial pour résoudre ce cas particulier.

Couplage

Borne inférieure

BIN-PACKING

- **Instance**. Entiers positifs ou nuls $a_1, \ldots, a_n$ et W.
- **Question**. Trouver un entier naturel k et une fonction $f : [n] \to [k]$ avec $\sum_{i:\, f(i)=j} a_i \leq W$ pour tout $j \in [k]$ tels que k soit minimum.

Donner une borne inférieure simple à la valeur de BIN-PACKING

$$\left\lceil \sum_{i=1}^{n} \frac{a_i}{W} \right\rceil \leq OPT$$

Borne inférieure

BIN-PACKING

- **Instance**. Entiers positifs ou nuls $a_1, \ldots, a_n$ et W.
- **Question**. Trouver un entier naturel k et une fonction $f : [n] \to [k]$ avec $\sum_{i:\, f(i)=j} a_i \leq W$ pour tout $j \in [k]$ tels que k soit minimum.

Donner une borne inférieure simple à la valeur de BIN-PACKING

$$\left\lceil \sum_{i=1}^{n} \frac{a_i}{W} \right\rceil \leq OPT$$

Formulation programme linéaire

Donner un PLNE modélisant le problème

- On a au plus K conteneurs ($K = n$ par exemple)

$x_{ij} = 1$ si objet i dans conteneur j
$= 0$ sinon

- $\rightarrow \sum_{i=1}^{n} x_{ij} a_i \leq W, \quad \forall j \in [\![1, K]\!]$

- $\sum_{j=1}^{K} x_{ij} = 1, \quad \forall i \in [\![1, n]\!]$

- $y_j = 1$ si il y a au moins 1 objet dans j
$= 0$ sinon

- Objectif : $\min_{x,y} \sum_{j=1}^{K} y_j$
- $\forall i,j \; x_{ij} \leq y_j$

Autre option : $\sum_j a_j x_{ij} \leq W y_j \quad \forall i$

Formulation programme linéaire

Donner un PLNE modélisant le problème

On suppose que l'on a k conteneurs disponibles.

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{k} z_j \\
\text{s.c.} \quad & \sum_{j=1}^{k} y_{ji} = 1 && i = 1, \ldots, n \\
& \sum_{i=1}^{n} a_i y_{ji} \leq W z_j && j = 1, \ldots, k \\
& y_{ji}, z_j \in \{0, 1\} && i = 1, \ldots, n;\ j = 1, \ldots, k
\end{aligned}
$$

où $z_j = 1$ si le conteneur j est utilisé et $y_{ji} = 1$ si l'objet i est mis dans le conteneur j.

Heuristiques

k - approximation

$OPT \leq SOL \leq k\, OPT$

1. NEXT-FIT : 2-approximation.
2. FIRST-FIT : $\frac{17}{10}$-approximation.
3. FIRST-FIT DECREASING : $\frac{3}{2}$-approximation.

Peut-on faire mieux ?

Heuristiques

1. NEXT-FIT : 2-approximation.
2. FIRST-FIT : $\frac{17}{10}$-approximation.
3. FIRST-FIT DECREASING : $\frac{3}{2}$-approximation.

Théorème. A moins que $P = NP$ il n'existe pas d'algorithme polynomial garantissant une approximation $\rho < \frac{3}{2}$

NEXT-FIT

On prend les objets les uns après les autres. Dès que l'objet i ne peut pas entrer dans le conteneur courant, je passe à un nouveau conteneur.

Théorème. NEXT-FIT fournit une solution SOL telle que

$$SOL \leq 2OPT - 1.$$

Sur quoi repose la preuve ?

- On a $\left\lceil \sum_{i=1}^{n} \frac{a_i}{W} \right\rceil \leq OPT$
- SOL :

$\sum a_i > W$

pour chaque paire de conteneurs

On a $\left\lfloor \frac{SOL}{2} \right\rfloor$ paires de conteneurs

$\sum_{i=1}^{n} a_i > W \left\lfloor \frac{SOL}{2} \right\rfloor$

$$\left\lfloor \frac{SOL}{2} \right\rfloor < \sum_i \frac{a_i}{W} \leq \left\lceil \sum \frac{a_i}{W} \right\rceil \leq OPT$$

$$\frac{SOL}{2} < OPT \quad (\text{car } OPT \text{ entier})$$

$$SOL < 2OPT$$

$$SOL + 1 \leq 2OPT \quad (\text{car } SOL \text{ et } 2OPT \text{ entiers})$$

NEXT-FIT

On prend les objets les uns après les autres. Dès que l'objet i ne peut pas entrer dans le conteneur courant, je passe à un nouveau conteneur.

Théorème. NEXT-FIT fournit une solution SOL telle que

$$SOL \leq 2OPT - 1.$$

Proof.

$$\forall j \leq \left\lfloor \frac{SOL}{2} \right\rfloor, \sum_{i:\ f(i)\in\{2j-1,2j\}} a_i > W \quad \text{donne} \quad \sum_{i=1}^{n} a_i > \left\lfloor \frac{SOL}{2} \right\rfloor W$$

et la borne inférieure $\left\lceil \sum_{i=1}^{n} \frac{a_i}{W} \right\rceil \leq OPT$ donne le résultat.

FIRST-FIT

Je prends les objets les uns après les autres. Je mets l'objet i dans le conteneur de plus petit rang où il peut entrer.

Théorème. FIRST-FIT fournit une solution SOL telle que

$$SOL \leq \left\lceil \frac{17}{10} OPT \right\rceil .$$

FIRST-FIT DECREASING

Je trie d'abord les objets par a_i décroissant. Puis j'applique FIRST-FIT.

Théorème. FIRST-FIT DECREASING fournit une solution SOL telle que

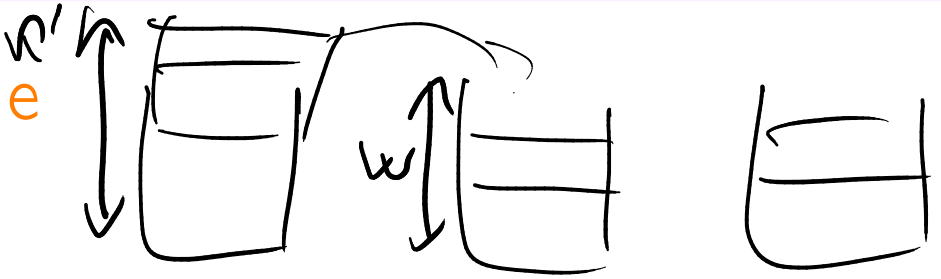
$$SOL \leq \frac{3}{2} OPT.$$

Quel avantage pratique ont NEXT-FIT et FIRST-FIT sur FIRST-FIT DECREASING ?

Quel avantage pratique ont NEXT-FIT et FIRST-FIT sur FIRST-FIT DECREASING ?

L'avantage de NEXT-FIT et de FIRST-FIT sur FIRST-FIT DECREASING est qu'ils peuvent fonctionner **online**.

Exemple de Recherche locale



On résout le problème de manière indirecte : on cherche à savoir s'il existe une solution en k conteneurs.

- Choisir $W' \geq W$, une capacité commune, pour laquelle il existe une solution réalisable.
- Appliquer une **recherche locale** pour minimiser W' : passage d'une solution à une solution voisine en changeant l'affectation d'un objet.
- Si à la fin $W' \leq W$: solution réalisable au problème de départ en k conteneurs.

On répète l'algorithme en faisant varier le nombre k.

Problème ouvert

Il existe encore de nombreuses instances non résolues : par exemple :

$W = 150$
$n = 120$
$a_i =$ 100 22 25 51 95 58 97 30 79 23 53 80 20 65 64 21 26 100 81 98 70 85 92 97 86 71 91 29 63 34 67 23 33 89 94 47 100 37 40 58 73 39 49 79 54 57 98 69 67 49 38 34 96 27 92 82 69 45 69 20 75 97 51 70 29 91 98 77 48 45 43 61 36 82 89 94 26 35 58 58 57 46 44 91 49 52 65 42 33 60 37 57 91 52 95 84 72 75 89 81 67 74 87 60 32 76 85 59 62 39 64 52 88 45 29 88 85 54 40 57

Meilleure solution connue : $k = 51$

1 Sac à dos

2 Bin Packing

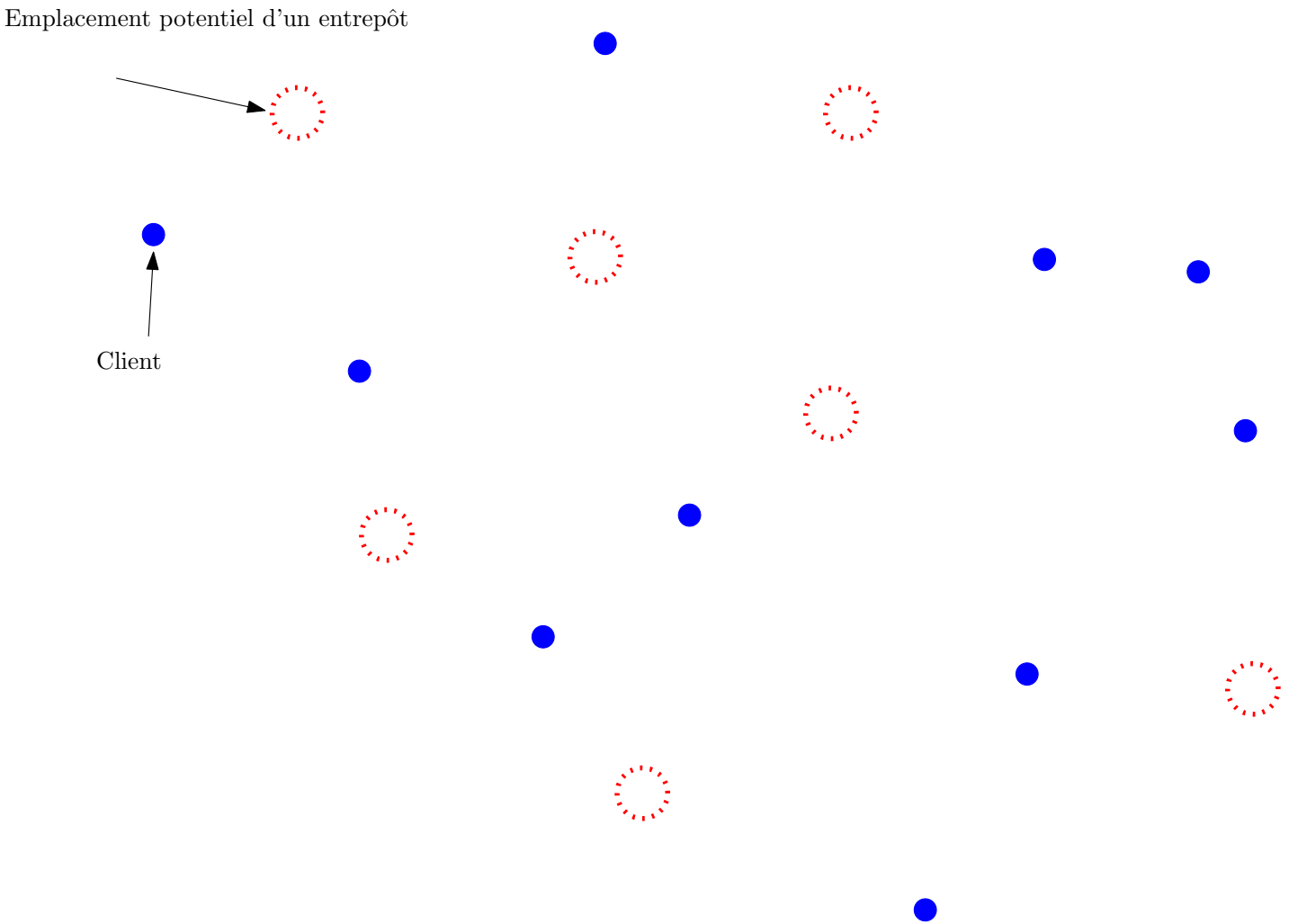
3 Positionnement d'entrepôts

4 Exercices

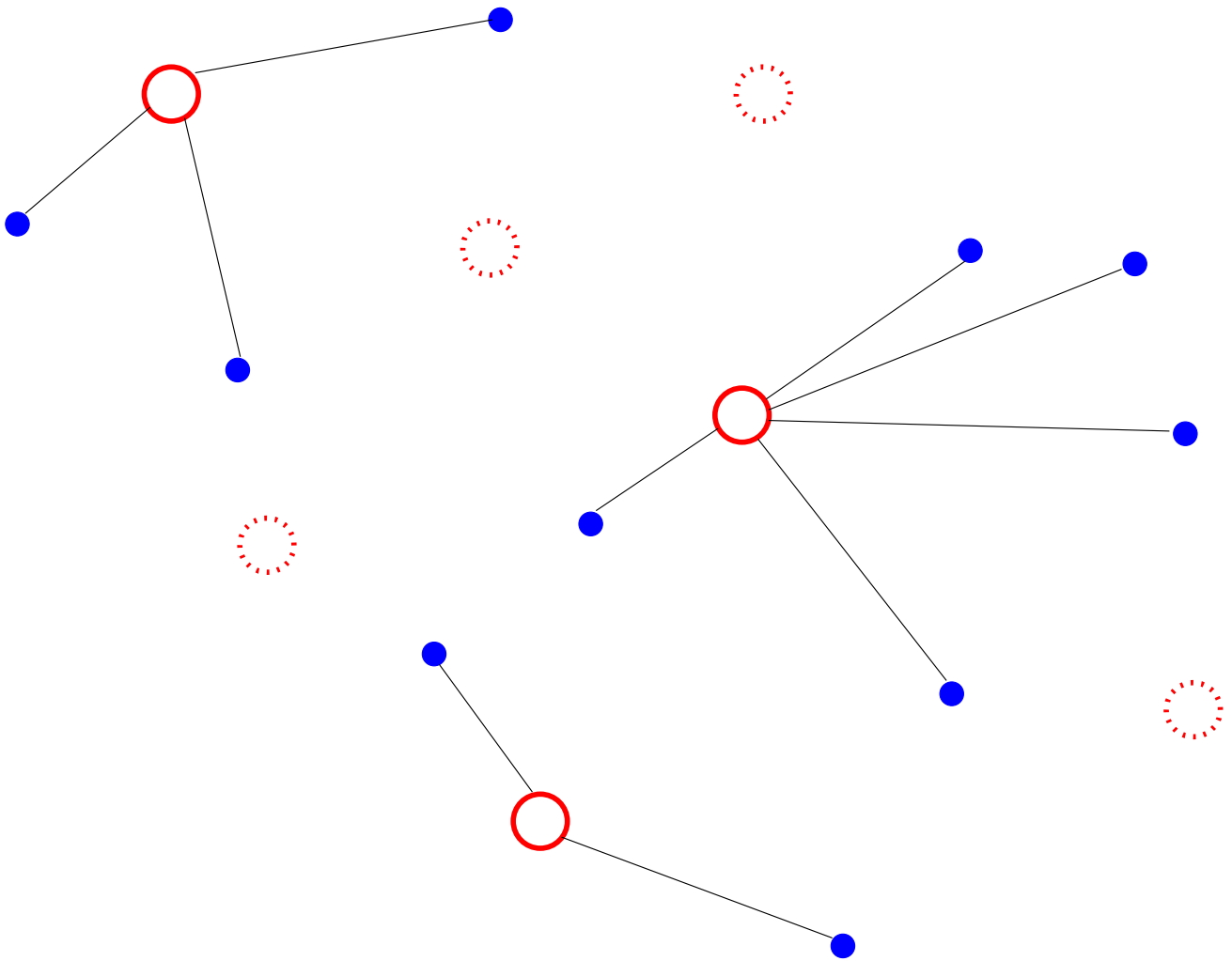
Présentation du problème

De nombreuses décisions économiques mettent en jeu la sélection ou le positionnement de dépôts, d'usines, de relais, d'hôpitaux, etc. afin de répondre de manière optimale à la demande.

Illustration



Illustration



Formalisation du problème

Positionnement d'entrepôts

- **Instance**. Clients $\mathcal{D}$, entrepôts potentiels $\mathcal{F}$, coût fixe $f_i \in \mathbb{R}_+$ d'ouverture pour entrepôt $i \in \mathcal{F}$, coût de service $c_{ij} \in \mathbb{R}_+$ pour $i \in \mathcal{F}$ et $j \in \mathcal{D}$.
- **Question**. Trouver $X \subseteq \mathcal{F}$ et affectation $\sigma : \mathcal{D} \to X$ tels que

$$\sum_{i \in X} f_i + \sum_{j \in \mathcal{D}} c_{\sigma(j)j}$$

soit minimale.

Complexité

Proposition. Le problème du positionnement d'entrepôts est **NP**-difficile.

Formulation sous forme PLNE

$j \in D$ $i \in F$

Donner un PLNE qui modélise le problème de positionnement d'entrepôts.

$x_i = \begin{cases} 1 \text{ si ouvre } i, \forall i \in F \\ 0 \text{ sinon} \end{cases}$ | $y_{ji} = 1$ client j connecté à i $= 0$ sinon

- $\min \underbrace{\sum_{i \in F} f_i x_i}_{\text{coût de construction}} + \underbrace{\sum_{i \in F} \sum_{j \in D} c_{ij} y_{ji}}_{\text{coût de service}}$

s.c. $\sum_{i \in F} y_{ji} = 1 , \forall j \in D$ (chaque client affecté à exactement un entrepôt)

$$y_{ji} \leq x_i \quad \forall i, j$$

$$x_i \in \{0, 1\}, \quad y_{ji} \in \{0, 1\}$$

Formulation sous forme PLNE

Donner un PLNE qui modélise le problème de positionnement d'entrepôts.

$$
\begin{aligned}
\text{Min} \quad & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} & \\
\text{s.c.} \quad & x_{ij} \leq y_i & i \in \mathcal{F},\ j \in \mathcal{D} \\
& \sum_{i \in \mathcal{F}} x_{ij} = 1 & j \in \mathcal{D} \\
& x_{ij} \in \{0,1\} & i \in \mathcal{F},\ j \in \mathcal{D} \\
& y_i \in \{0,1\} & i \in \mathcal{F}.
\end{aligned}
$$

Recherche locale

1. Comment encoder une solution ? $\rightarrow (y_i)_{i \in F}$
2. Quel voisinage proposez vous ?

Recherche locale

1. Comment encoder une solution ?
2. Quel voisinage proposez vous ?

1. Le voisinage naturel est basé sur la remarque suivante :
   L'espace des solutions peut être identifié à l'ensemble des

   parties X de $\mathcal{F}$.

Recherche locale

1. Comment encoder une solution ?
2. Quel voisinage proposez vous ?

1. Le voisinage naturel est basé sur la remarque suivante : *L'espace des solutions peut être identifié à l'ensemble des parties X de $\mathcal{F}$.*
2. Modifications locales efficaces : passer de X à X' avec
   - $X' := X \setminus \{x\}$ pour un $x \in X$, (**drop**)
   - $X' := X \cup \{x'\}$ pour un $x' \in \mathcal{F} \setminus X$ (**add**) ou
   - $X' := X \setminus \{x\} \cup \{x'\}$ (**swap**).

Exercice

$\forall i \in \mathcal{F}, \sum_{j \in \mathcal{D}} x_{ji} \leq M_i \, y_i$

On reprend le problème de positionnement d'entrepôts vu en cours, mais avec cette fois une contrainte supplémentaire : chaque entrepôt ne peut desservir qu'un nombre limité de clients. Modéliser le problème suivant comme un problème linéaire en nombres entiers.

Données : Un ensemble fini de clients $\mathcal{D}$, un ensemble fini d'entrepôts potentiels $\mathcal{F}$, un coût fixe $f_i \in \mathbb{R}_+$ d'ouverture et une capacité K_i pour chaque entrepôt $i \in \mathcal{F}$, et un coût de service $c_{ij} \in \mathbb{R}_+$ pour chaque $i \in \mathcal{F}$ et $j \in \mathcal{D}$.

Demande : Trouver un sous-ensemble $X \subseteq \mathcal{F}$ (dits *entrepôts ouverts*) et une affectation $\sigma : \mathcal{D} \to X$ des clients aux entrepôts ouverts, tel que pour tout i, l'entrepôt i ne desserve pas plus de K_i clients, de façon à ce que la quantité $\sum_{i \in X} f_i + \sum_{j \in \mathcal{D}} c_{\sigma(j)j}$ soit minimale.

Correction

$$
\begin{aligned}
\text{Min} \quad & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} & \\
\text{s.c.} \quad & \sum_{j \in \mathcal{D}} x_{ij} \leq K_i y_i & i \in \mathcal{F} \\
& \sum_{i \in \mathcal{F}} x_{ij} = 1 & j \in \mathcal{D} \\
& x_{ij} \in \{0,1\} & i \in \mathcal{F},\ j \in \mathcal{D} \\
& y_i \in \{0,1\} & i \in \mathcal{F}.
\end{aligned}
$$

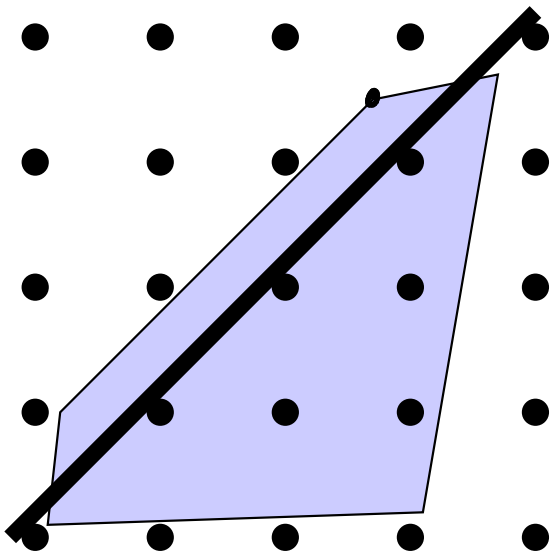
1 Sac à dos

2 Bin Packing

3 Positionnement d'entrepôts

4 Exercices

Inégalité valide : rappel



Une inégalité valide n'a d'intérêt que si elle réduit la taille du polytope !

Inégalité valide pour le sac à dos

On considére un problème de sac-à-dos avec $a_1, \ldots, a_n, b$ des réels positifs. On note S l'ensemble des solutions réalisables

$$S := \left\{ \boldsymbol{x} \in \{0,1\}^n : \sum_{i=1}^{n} a_i x_i \leq b \right\}.$$

On appelle *ensemble dépendant* tout ensemble $C \subseteq \{1, \ldots, n\}$ tel que $\sum_{i \in C} a_i > b$.

1. Montrer que tout $\boldsymbol{x} \in S$ satisfait l'inégalité

$$\sum_{i \in E(C)} x_i \leq |C| - 1,$$

où $E(C) = C \cup \{j : a_j \geq \max_{i \in C} a_i\}$.

2. Est-ce le cas pour des solutions du relâché continu ?
3. Supposons que l'on résolve le relâché continu : comment identifier l'ensemble C pour lequel la contrainte est la plus violée ? A quoi cela sert-il ?

$S = \{ x \mid \sum_{i=1}^{n} a_i x_i \leq b \}$ $\quad E(C) = C \cup \{ j \mid a_j \geq \max_{i \in C} a_i \}$

$C \subseteq [\![1, n]\!]$ tq $\sum_{i \in C} a_i > b$

1) $x \in S$

Sq $\sum_{i \in E(C)} x_i \geq |C|$

$$\sum_{i=1}^{n} a_i x_i = \sum_{i \in E(C)} a_i x_i + \sum_{i \notin E(C)} a_i x_i$$

$$= \sum_{i \in C} a_i x_i + \sum_{i \in E(C) \setminus C} a_i x_i + \underbrace{\sum_{i \notin E(C)} a_i x_i}_{\geq 0}$$

$$\geq \text{———} + \text{——} \max_C a_i \text{—}$$

$$\geq \sum_{i \in C} a_i > b$$

2) $b = 3$, $a_1 = 2$, $a_2 = 2$

$x_1 = 1$, $x_2 = 0,5$

$a_1 x_1 + a_2 x_2 \leq b$

$C = \{1, 2\}$

$x_1 + x_2 = 1,5 > |C| - 1$

3) Problème de séparation: (x fixé $\in [0,1]^n$)

$$\begin{cases} \min_C |C| - 1 - \sum_{i \in E(c)} x_i \\ \lambda . c \quad \sum_{i \in C} a_i > b \end{cases}$$

Correction

1. C n'est pas faisable, donc au plus $|C| - 1$ éléments sont sélectionnés à l'intérieur.
2. Par exemple, si l'on deux items de poids $a_i = 2$, une capacité $W = 3$, et $C = \{1, 2\}$, alors la solution du relâché linéaire $x_1 = 1$ et $x_2 = 1/2$ viole l'inégalité valide.
   De manière générale, il suffit par exemple de prendre la solution optimale du relâché continu pour avoir un contre exemple.

   (presque sûrement par rapport à la mesure uniforme sur les instances).
3. Par exemple par programmation dynamique (ou par un problème de plus court chemin sous-contrainte.)

Inégalité valide pour le bin-packing

On rappelle la modélisation PLNE du problème du bin-packing. On suppose que l'on dispose d'un stock de K boîtes de taille 1.

$$\begin{array}{lll} \min & \sum_{j=1}^{K} z_j & \\ \text{s.c.} & \sum_{j=1}^{K} y_{ji} = 1 & \text{pour } i = 1, \ldots, n \\ & \sum_{i=1}^{n} a_i y_{ji} \leq z_j & \text{pour } j = 1, \ldots, K \\ & y_{ji}, z_i \in \{0, 1\} & \text{pour } i = 1, \ldots, n; \\ & & j = 1, \ldots, K \end{array}$$

où $z_j = 1$ si la boîte j est utilisée et $y_{ji} = 1$ si l'objet i est mis dans la boîte j.

1. Montrer qu'en ajoutant les contraintes $z_j \geq z_{j+1}$, pour $j = 1, \ldots, K-1$, on continue à modéliser le problème.
2. Même question avec la contrainte $\sum_{j=1}^{K} z_j \geq \lceil \sum_{i=1}^{n} a_i \rceil$.
3. Expliquer l'intérêt de ce type d'inégalité dans une résolution d'un problème de bin-packing par branch-and-bound.

Correction

1. Cela oblige juste à utiliser les premiers conteneurs, mais n'impacte pas la valeur de la solution optimale.
2. Borne inférieure que l'on a donné.
3. Briser la symétrie pour la premiére (très important) : divise par $\binom{K}{OPT}$ la taille de l'ensemble des solutions optimales.
   Restreindre la taille du polytope pour la seconde.

Relaxation Lagrangienne pour le bin packing

On rappelle le PLNE pour le bin-packing

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{k} z_j \\
\text{s.c.} \quad & \sum_{j=1}^{k} y_{ji} = 1 && i = 1, \ldots, n \\
& \sum_{i=1}^{n} a_i y_{ji} \leq W z_j && j = 1, \ldots, k \\
& y_{ji}, z_j \in \{0, 1\} && i = 1, \ldots, n;\ j = 1, \ldots, k
\end{aligned}
$$

1. Procéder à la relaxation Lagrangienne de $\sum_{j=1}^{k} y_{ji} = 1$
2. Montrer que la borne $z_{RL}(\lambda)$ se calcule facilement (quel problème doit on résoudre à chaque fois)
3. Montrer que la borne donnée par la relaxation lagrangienne est meilleure que la borne donnée par la relaxation linéaire

Relaxation Lagrangienne pour le bin packing

La correction est dans le nouveau poly, Section 11.3.1