Bin Packing

20 décembre

Le problème dit BinPacking est défini ainsi :

Instance : un entier naturel C et une famille $X = x_0, \ldots, x_{n-1}$ d'entiers naturels

Solution admissible : une partition de X en $B_0 \sqcup \cdots \sqcup B_{k-1}$ telle que $\sum_{x \in B_i} x \leq C$ pour tout i

Optimisation : minimiser k

Autrement dit, on nous donne n *objets* de *volumes* $x_0, \ldots, x_{n-1}$, et l'on dispose de *boîtes* de *capacité* C. Il faut répartir les objets dans k boîtes de manière à ce que la somme des volumes reste toujours inférieure à la capacité, en minimisant le nombre k de boîtes utilisées.

1 Donner une solution optimale pour BinPacking sur l'instance suivante : $C = 10$ et $X = 2, 5, 4, 7, 1, 3, 8$.

0.1 Caractère NP-complet

On admet (*cf* exercice TD réductions) que le problème SubsetSum, défini comme suit, est NP-complet :

Instance : un multi-ensemble A d'entiers naturels et un entier $s > 0$

Question : existe-t-il $B \subseteq A$ tel que $\sum_{x \in B} x = s$.

2 Définir le problème de décision BPD associé au problème d'optimisation BinPacking.

3 On considère le problème Partition :

Instance : n entiers naturels $x_0, \ldots, x_{n-1}$

Question : Existe-t-il $I \subseteq [0 \ldots n-1]$ tel que $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$?

Montrer que Partition est NP-complet. *On pourra partir d'une instance de* SubsetSum *et lui ajouter* $2s$ *et* $\sum_{i=0}^{n-1} x_i$.

4 En déduire que BPD est NP-complet.

0.2 Stratégies gloutonnes

On va considérer dans ce sujet trois stratégies gloutonnes pour le problème BinPacking. On suppose à partir de maintenant que les poids des différents objets sont majorés par C, puisque sinon il n'y a clairement aucune solution.

- La stratégie **next-fit** considère les objets dans l'ordre d'arrivée, et ajoute ces objets dans la boîte courante tant que c'est possible. Quand ce n'est plus le cas, on ferme (définitivement) cette boîte et l'on en crée une nouvelle, qui devient la boîte courante.

- La stratégie **first-fit** considère aussi les objets dans l'ordre d'arrivée, mais maintient une liste (initialement vide) de boîtes $B_0, \ldots, B_{k-1}$. À chaque fois que l'on considère un objet, on cherche le premier i tel que l'objet rentre dans la boîte B_i :
  - s'il en existe un, on ajoute l'objet dans cette boîte ;
  - sinon, on crée une nouvelle boîte B_k dans laquelle on place l'objet.
- La stratégie **first-fit-decreasing** procède comme **first-fit** mais commence par trier les objets par ordre décroissant de volume.

0.3 Analyse de *next-fit*

On note m le nombre de boîtes utilisées par la stratégie *next-fit* sur une instance $C, X = (x_0, \ldots, x_{n-1})$ et m^* le nombre optimal de boîtes sur cette même instance. On numérote B_0, B_{m-1} les boîtes utilisées par *next-fit*, dans l'ordre de leur utilisation et l'on note v_i le volume total des objets présents dans la boîte v_i et $V = \sum_{i=0}^{n-1} x_i$.

5 Montrer que $v_i + v_{i+1} > C$ pour $0 \leq i < m - 1$.

6 En déduire que *next-fit* fournit une 2-approximation pour BinPacking.

0.4 Analyse de *first-fit-decreasing*

On reprend les notations de la partie précédente, avec m le nombre de boîtes utilisées par *first-fit-decreasing* et m^* le nombre optimal de boîtes. On note $x_0 \geq \cdots \geq x_{n-1}$ les poids.

On considère la boîte B_j avec $j = \lfloor \frac{2m}{3} \rfloor$, et l'on note x le volume maximal d'un objet rangé dans la boîte B_j.

Cas $x > C/2$

7 Montrer que $m \leq \frac{3}{2} m^*$.

Cas $x \leq C/2$ On note v le plus grand volume disponible (en fin d'exécution) dans les boîtes $B_0, \ldots, B_{j-1}$.

8 Montrer que $\sum_{l=j}^{m-1} v_l > v + 2v(m - j - 1)$.

9 Conclure.

0.5 Difficulté de l'approximation

10 Soit $\epsilon > 0$. Par réduction de Partition, montrer que s'il existe un algorithme fournissant en temps polynomial une $(\frac{3}{2} - \epsilon)$-approximation pour BinPacking, alors $P = NP$.