Denis TRYSTRAM

Algorithmique Avancée

ENSIMAG 2A - filière Apprentissage

Bin Packing

1 Les fous du rangement...

Le problème décrit ci-dessous est connu sous le nom de *BinPacking* (que l'on pourrait traduire par rangement dans des boites).

On considère un ensemble de n objets de tailles s_i pour $1 \leq i \leq n$ et un grand nombre de "boites" de hauteur H. On veut ranger les objets en les empilant dans un nombre minimal de boites. Bien sûr, on ne pourra pas dépasser la capacité H de chaque boite. Dans tout le problème, pour une instance donnée on notera N_A le nombre de boites obtenu en appliquant l'algorithme A. N_{opt} désigne le nombre optimal de boites.

Ce problème s'exprime sous la forme suivante :

BinPacking
Input : n objets (de tailles entières s_i), un entier H (la hauteur maximal des boites).
Question : Déterminer un empilement des objets dans un nombre minimum de boites, sans dépasser la capacité de chaque boite.

1.1 Complexité

Formuler ce problème sous forme de décision.

Sous forme de décision, il suffit d'introduire un entier N comme ci-dessous :
BinPacking décision
Input : n objets (de tailles entières s_i), un entier H (la hauteur maximal des boites) et un entier N.
Question : Est-il possible d' empiler les n objets dans N boites (sans dépasser la capacité de chaque boite) ?

Vérifier que ce problème est dans $\mathcal{NP}$.

La question ici est de construire un *certificat d'authentification* polynomial qui vérifie qu'une solution est valide.

Pour la version de décision du problème, il est facile de vérifier que toutes les boites dans cette solution sont bien remplies à moins de H et de compter le nombre de boites. Si ce nombre est plus petit que N, la réponse est *OUI*.

Cette procédure est évidemment polynomiale (plus précisément, elle est dans $\mathcal{O}(n)$).

Montrer que ce problème est NP-complet.

Comme le problème appartient à $\mathcal{NP}$, il suffit de trouver un problème NP-complet auquel réduire notre problème. On doit donc trouver ce problème et construire la réduction.

On a deux candidats naturels ici : soit 2partition, soit KnapSack. Montrons la réduction à partir de 2partition :

2partition
Input : n entiers n_i dont la somme est paire.
Question : Est-il possible de partitioner les n entiers en deux parties égales A_1 et A_2, c'est-à-dire tels que $\Sigma_{i \in A1} n_i = \Sigma_{i \in A2} n_i$?

La réduction est la suivante : on transforme une instance quelconque de 2partition de la façon suivante en instance de BinPacking :

$s_i = n_i$ et $H = \frac{\Sigma_i n_i}{2}$.

Il est facile alors de vérifier qu'une instance de ce type de BinPacking est positive si et seulement si l'instance initiale de 2partition l'est.

Nous allons étudier dans la suite deux cas particuliers.

1.2 Petits objets

Montrer rapidement que l'on peut se ramener à un problème où les boites sont de tailles unitaires et des objets rationnels.

C'est immédiat en transformant chaque s_i en $\frac{s_i}{H}$.

L'hypothèse ici est que la taille maximal d'un objet est ρ ($0 < \rho \leq 1$).

On propose d'analyser l'heuristique FF (First Fit) suivante : on trie les objets par tailles décroissantes, et on range les objets l'un après l'autre dans la première boite possible où il rentre.

1. **Cas 1.** $\rho \geq \frac{1}{2}$.

   Par un argument de borne inférieure, montrer que $N_{FF} \leq 2.N_{opt} + 1$

   La borne inf en question (notée LB) est obtenue par le remplissage fluide des boites (jusqu'en haut, en coupant une tâche si besoin), aucun espace n'est laissé vide sauf éventuellement dans la dernière boite.

2. **Cas 2.** $\rho \leq \frac{1}{2}$.

   Montrer dans ce cas : $N_{FF} \leq (1 + 2\rho)N_{opt} + 1$[1].

[1] On utilisera l'identité suivante (après l'avoir justifié) : $\frac{1}{1-x} \leq 1 + 2x$ pour $0 \leq x \leq \frac{1}{2}$

La preuve ici est obtenue en comparant la politique fluide à FF. L'espace laissé libre dans chaque boite est au plus $1-\rho$ sauf dans la dernière boite, ainsi :

$(N_{FF}-1)(1-\rho) \leq LB \leq N_{opt}$

$N_{FF} \leq 1 + \frac{1}{1-\rho} N_{opt}$

On utilise l'inégalité donnée pour $x = \rho$ pour obtenir le résultat.

1.3 Nombre limité d'objets

On suppose ici qu'il y a au maximum k tailles différentes (plusieurs objets ayant la même taille). On a n_i objets du type i, et donc, $\Sigma_{1 \leq i \leq k} = n$.

L'idée est de dénombrer tous les sous-ensembles d'objets possibles et de déterminer le nombre optimal de bins pour chaque sous-ensemble.

Déterminer un algorithme polynomial pour générer tous les sous-ensembles possibles.

On note Q l'ensemble de toutes les configurations possibles. Clairement, on peut coder les configurations par des k-uplets $(q_1, q_2, ..., q_k)$ où $0 \leq q_1 \leq n_1$, etc.. Le nombre de configurations possibles est $(n_1+1)(n_2+1)...(n_k+1)$ qui est dans $\mathcal{O}(n^k)$ car chaque $n_i \leq n$.

Construire une solution exacte en utilisant la programmation dynamique et donner une estimation asymptotique du coût.

Notons $N^*(q)$ le nombre minimum de bins nécessaires pour empiler les objets de la configuration $q = (q_1, q_2, ..., q_k)$.

On calcule les éléments d'un tableau k-dimensionnel pour chaque k-uplet $(q_1, q_2, ..., q_k) \in \{0, ..., n_1\} \times ... \times \{0, ..., n_k\}$.

Ce tableau est initialisé avec $N^*(q) = 1$ pour $q \in Q$.

On utilise ensuite la relation de récurrence suivante :

$N^*(q_1, q_2, ..., q_k) = 1 + min_Q(N^*(q_1 - \alpha_1, q_2 - \alpha_2, ..., q_k - \alpha_k))$.

Calculer chaque entrée nécessite au plus $\mathcal{O}(n^k)$.

Ainsi, pour calculer $N^*(n_1, n_2, ..., n_k)$ l'algorithme demandera $\mathcal{O}(n^{2k})$ opérations.

On pourra dérouler l'algorithme sur l'exemple suivant :

$n = 6$, $k = 2$, les objets étant répartis en 4 objets de tailles 0.3 et 2 objets de tailles 0.4.

Il y a 15 configurations possibles.