Aix-Marseille Université – Licence Informatique – L2

UE Algorithmique 1

TP 4 – Sac à dos

Etant donné un ensemble P de n objets de poids $p_1 \cdots p_n$ et de valeurs $v_1 \cdots v_n$, et un sac à dos de capacité c, le problème du sac à dos consiste à trouver un sous-ensemble S de P tel que :

$$\sum_{i \in S} p_i < c \text{ et } \sum_{i \in S} v_i \text{ est maximal}$$

Il faut donc choisir parmi les objets ceux que l'on va emporter. Ce problème est NP-complet, on ne connaît pas d'algorithme polynomial pour le résoudre. Dans ce TP vous allez implémenter deux approches très différentes (qui ne conduisent pas à des algorithmes polynomiaux).

Énumération exhaustive

L'idée la plus simple consiste à explorer par un algorithme de *backtracking* l'espace de tous les sous-ensembles de P qui satisfont la contrainte de capacité et de sauvegarder la solution de plus grande valeur trouvée.

Écrivez une fonction qui implémente cette idée en construisant tous les sous-ensembles de P dont la somme des poids est au plus la capacité du sac. Vous essaierez de rendre cette approche plus efficace en détectant le plus tôt possible, pendant l'exploration, les configurations sans intérêt (i.e. qui ne pourront conduire à une meilleure solution que la meilleure solution rencontrée jusqu'à présent). Réordonner les éléments par ratio valeur sur poids croissant permet d'obtenir plus rapidement une bonne solution lors de l'exploration et donc d'éliminer plus tôt les configurations qui ne peuvent pas conduire à une meilleure solution.

Approche de type programmation dynamique

La programmation dynamique permet d'obtenir un algorithme pseudo-polynomial (polynomial en le nombre d'entrées et en le plus grand des nombres apparaissant dans ces entrées). On utilise une table A à deux dimensions dans la quelle $A(i,\, p)$ représente le profit maximal d'objets de $\{1 \cdots i\}$ dont le poids total est inférieur ou égal à p. $A(i,\, p)$ vaut 0 si il n'existe aucun sous-ensemble de $\{1 \cdots i\}$ de poids au plus p. La formule récursive suivante permet de calculer la table A :

$A(i+1,\, p) = max(A(i,\, p),\, A(i,\, p - p_{i+1}) + v_{i+1})$ si $p_{i+1} \leq p$

$A(i+1,\, p) = A(i,\, p)$ sinon

Si $p_{i+1} > p$ alors on ne peut pas prendre l'objet $i+1$, donc $A(i+1,\, p) = A(i,\, p)$. Sinon, on a deux choix. Si on ne prend pas l'objet $i+1$, $A(i+1,\, p) = A(i,\, p)$ et si on prend l'objet $i+1$ alors $A(i+1,\, p) = A(i,\, p - p_{i+1}) + v_{i+1}$ En prenant la meilleure des deux solutions précédentes, on est certain de calculer correctement $A(i+1,\, p)$.

Précisez les dimensions de la table. Vous écrirez une fonction qui calcule le meilleur chargement en calculant les valeurs de la table comme indiqué ci-dessus, et une fonction qui calcule un chargement optimal (il peut y en avoir plusieurs) à partir de la table.

Indication pour l'extraction du chargement : partir de la valeur $A(i,\, p)$ la plus grande (et inférieure à c), on retrouve les choix effectués en comparant $A(i,\, p)$ avec $A(i-1,\, p)$: s'ils sont égaux, un choix (possible) est de ne pas prendre l'objet i, sinon on doit nécessairement le prendre.