TD/TP Problèmes NP

Dans ce TP on se propose d'examiner différentes manières de coder et résoudre des problèmes de la classe NP. on s'intéresse au problème du sac à dos. On suppose que l'on dispose de i items ayant chacun une utilité (ou gain) u_i. Chacun de ces items a un poids m_i. On cherche à maximiser le gain en empaquetant le plus d'items possibles dans un sac de capacité maximale M. On distingue deux cas intéressants *i)* les items ne sont disponibles qu'en un seul exemplaire, i.e. on cherche à déterminer la quantité $x_i \in \{0, 1\}$ associée à chaque item, *ii)* on peut prendre plusieurs fois le même item, i.e. $x_i \in \mathbb{N}^+$. Le problème se formalise de la manière suivante:

$$U = \max_{x_i} \sum_i x_i u_i \quad (1)$$

$$\text{s.c.} \sum_i x_i m_i \leq M \quad (2)$$

On va examiner différentes méthodes pour résoudre ce problème vous permettant de sentir sa complexité. Vous générez pour chaque test que vous ferez un vecteur d'utilité et un vecteur de poids qui seront des entiers tirés aléatoirement dans $[1, 10]$. Vous réglerez M en fonction du nombre d'items possibles, par exemple si vous avez à votre dispositions n items (qui sera un paramètre de votre procédure de test), vous pourrez choisir $M = 7n$. Vous écrirez une fonction SOLVE_BAG pour chaque variante qui prendra les vecteurs d'utilité, de poids et M en paramètres et rendra la valeur max (gain total) atteinte, ainsi que le temps lié au calcul.

1 Approche force brute

1. On ne s'embarasse pas de considérations complexes ici : écrire une méthode qui calcule toutes les combinaisons possible (2^n !), les évalue, et renvoie le gain optimal.

2. même chose mais cette fois ci on peut choisir plusieurs fois le même item. On pourra determiner, pour chaque item la borne max du nombre de fois où on peut choisir cet item comme la partie entière de M/m_i. Attention, les temps de calcul peuvent devenir très long pour des valeurs de M élevés.

2 Approche gloutonne

On calcule pour chaque objet le rapport gain/masse (u_i/m_i). On trie les objets par ordre décroissant, puis on remplit le sac dans cet ordre jusqu'à ne plus pouvoir rajouter d'items. Comparer la qualité de la solution obtenue avec celle du solveur exact précédent. Trouvez notamment des cas de figure où la stratégie gloutonne ne donne pas la solution optimale du problème. Ici encore vous coderez deux versions de la fonction (un seul item disponible et nombre illimité d'items à disposition).

3 Programmation dynamique

On se limite ici au cas où un seul item de chaque objet est disponible.

L'idée de la programmation dynamique est de résoudre incrémentalement des versions plus simples du problème, et de stocker des résultats intermédiaires nécessaires pour ajouter de nouvelles variables. On réalise alors un compromis temps/espace. Dans le cas du problème du sac à dos, le problème est dit à *sous-structure optimale*, c'est à dire qu'on peut trouver la valeur optimale du problème à i variable à

partir de la valeur optimale à $i-1$ variables. On définit par récurrence la quantité $P(k,m)$ suivante, décrivant l'état du système pour k variables :

$$P(k,m) = \max_{x_i} \sum_{i}^{k} x_i u_i \quad (3)$$

$$\text{s.c.} \sum_{i} x_i m_i \leq m \quad (4)$$

alors la solution optimale est soit :

1. la solution optimale $P(k-1,m)$ où l'on choisit de ne pas rajouter l'item, i.e. $x_k = 0$
2. la solution optimale $P(k-1, m-m_k) + u_k$ où l'on choisit de rajouter l'item, i.e. $x_k = 1$

Il suffit alors de construire un tableau des différentes possibilités $P(k,m)$. Une fois ce tableau construit, il suffit de partir de la case $P(k,M)$ et de remonter à la case $P(0,.)$ pour savoir si l'on choisit l'item ou non et construire ainsi la solution.

On note alors que la complexité de l'algorithme est en temps et en espace $o(nM)$. Bien que d'allure polynomiale, on n'a pas montré que $P = NP$: le codage de M se faisant sur $log(M)$ bits, on reste bien en **complexité exponentielle** de la taille de l'entrée.

Coder et tester cet algorithme.

4 Bilan

Pour chaque méthode, faire varier le nombre d'items n, et mesurez un temps moyen pris sur 10 résolution du problème. Tracez les courbes de temps d'exécution moyen correspondantes pour les 3 méthodes dans les 2 cas de figure (une seule ou plusieurs fois le même item, où on n'exclura la programmation dynamique).