Examen d'optimisation combinatoire
Durée 2h, avril 2010
Le barème est indicatif

Exercice 1 Méthode arborescente (6 points)

Dans votre projet, vous avez traité le problème d'une méthode arborescente pour le problème du sac à dos inversé. Etant donnés n objets, l'objet i étant caractérisé par un poids p_i et une utilité w_i. On se donne une valeur W, et l'on cherche à déterminer un sous ensemble d'objets de poids minimum dont la somme des utilités est au moins W.

1 Indiquez ici les principaux éléments de la méthode arborescente que vous avez choisie (Définition des noeuds/solutions, évaluation par défaut, par excès, parcours de l'arbre).

La question devait traiter des choix que vous aviez vous-même effectués. Mais nous vous donnons ici le corrigé d'une telle méthode.

Pour définir une telle méthode, il convient de définir les éléments suivants :

1. *Définition des noeuds.* Comme pour le problème du sac à dos traditionnel, on peut utiliser pour chaque noeud S un ensemble d'objets choisis $C(S)$ et un ensemble d'objets rejetés $R(S)$. L'ensemble des solutions du problème associées à un noeud est l'ensemble des sacs qui rapportent au moins W et qui contiennent tous les objets de $C(S)$ et aucun objet de $R(S)$. Dans la suite on notera $V(S)$ l'ensemble des objets variables (ceux qui ne sont ni dans $C(S)$ ni dans $R(S)$).
2. *Définition de la séparation d'un noeud.* Comme pour le sac à dos traditionnel, pour séparer un noeud S on choisit un objet j de $V(S)$ et on crée deux fils S' et S'' avec

$$C(S') = C(S) \cup \{j\}, \quad R(S') = R(S), \quad C(S'') = C(S), \quad R(S'') = R(S) \cup \{j\}$$

3. *Définition de l'évaluation par défaut.* Il s'agit ici d'un problème de minimisation. Il faut donc trouver un minorant de toutes les solutions du noeud, ce qui se fait par relaxation du problème. Si la somme des profits des objets de $C(S)$ et de $V(S)$ est inférieure à W alors il n'y a aucune solution réalisable dans S. On définit donc $g(S) = +\infty$. Sinon, on peut considérer le problème relâché où tous les objets sont prédécoupés en éléments de profit 1 et où l'on a le droit de mettre des morcaux d'objets dans le sac. Ainsi, l'objet i sera constitué de w_i morceaux de poids $\frac{p_i}{w_i}$. Dans le cas où chaque objet est de profit 1,

le poids minimum d'un sac qui rapporte W consiste à choisir les W objets de plus petits poids. Si l'on cherche une telle solution pour le problème avec objets pré-découpés, cela consistera à commencer par mettre les objets de $C(S)$ dans le sac puis à trier les objets de $V(S)$ par $\frac{p_i}{w_i}$ croissants (et donc par $\frac{w_i}{p_i}$ décroissants) et à les mettre dans le sac un à un jusqu'à ce qu'on atteigne ou dépasse un profit W. On peut ensuite, si un objet dépasse, n'en prendre que la fraction qui permet d'atteindre W. La partie entière supérieure du poids de ce sac est un minorant de la solution optimale qu'on définit comme la valeur de $g(S)$ (puisqu'il s'agit d'une relaxation).

4. *Définition de l'évaluation par excès* Il s'agit ici de définir un majorant de la solution optimale d'un noeud. Il suffit de construire une solution réalisable (s'il en existe une). On peut s'appuyer sur l'algorithme précédent : Après avoir trié les objets, et mis les objets de $C(S)$ dans le sac, on rentre les objets de $V(S)$ un à un jusqu'à atteindre ou dépasser W. Si l'on atteint pas W, c'est que $h(S) = g(S) = +\infty$ (il n'y a pas de solution réalisable). Sinon, on considère le poids des objets de ce sac qui définit une solution réalisable, et donc la valeur de $h(S)$. C'est également cette valeur qui permet, au cours de la méthode arborescente, de mettre à jour la meilleure solution rencontrée X_0.
5. *Rappel des règles de troncature :* Dans le cas d'un problème de minimisation, un noeud sera tronqué si $f(X_0) \leq g(S)$ ou si $g(S) = h(S)$.
6. *Règles de parcours de l'arbre et choix de l'objet de séparation* On peut par exemple proposer un parcours par meilleure évaluation par défaut d'abord et de choisir l'objet découpé dans l'évaluation par défaut comme objet de séparation.

2 Utilisez cette méthode arborescente pour résoudre l'exemple suivant :

$W = 45$

i	1	2	3	4	5	6	7
p_i	3	4	5	4	2	5	3
w_i	15	18	20	12	6	20	15

En triant les objets par w_i/p_i décroissant on observe que nous avons dans l'ordre les objets $1, 7, 2, 3, 6, 4, 5$.

A la racine de l'arbre S_0 on a $C(S_0) = R(S_0) = \emptyset$. Pour calculer l'évaluation par défaut, on met dans le sac les objets $1, 7$ et $15/18 = 5/6$ de l'objet 2. Cela nous donne donc un poids de $\lceil 6 + 4 * 5/6 \rceil = 10 = g(S_0)$.

L'évaluation par excès consiste à mettre dans le sac les objets $1, 7, 2$ ce qui pèse également $10 = h(S_0)$. On en déduit que c'est la solution optimale du problème posé.

Exercice 2 Deux machines(11 points)

Un atelier de production dispose de deux machines qui peuvent fonctionner en parallèle. En début de mois un ensemble de commandes (tâches) est proposé à l'atelier pour réalisation. La plupart du temps, il y en a trop pour être réalisées. Le responsable d'atelier doit donc décider quelles sont les commandes qui sont acceptées par son atelier (celles qui peuvent être réalisées dans le mois), et sur quelle machine elles s'effectuent. Pour une tâche i la durée n'est pas la même selon la machine qu'on utilise (durées respectives a_i et b_i). De plus, le fait d'accepter une tâche i induit un profit w_i pour l'entreprise qui peut la facturer au client. On cherche donc à ordonnancer les tâches sur les deux machines de sorte que la durée des tâches sur chacune des machine soit inférieure à la durée ouvrable du mois M en maximisant le profit de l'atelier.

1 Modéliser ce problème à l'aide d'un programme mathématique.

Notons n le nombre de tâches. On définit d'abord les inconnues : x_i vaut 1 si la tâche i est acceptée et faite sur la machine 1, 0 sinon. y_i vaut 1 si la tâche i est acceptée et faite sur la machine 2, 0 sinon. A noter que si une tâche est acceptée elle ne peut être faite que sur l'une des machines. Par conséquent on a pour toute tâche i,

$$x_i + y_i \leq 1$$

Le fait qu'une tâche i soit acceptée se représente donc par la quantité $x_i + y_i$. Notre objectif est de maximiser le profit réalisé, c'est à dire

$$\sum_{i=1}^{n} w_i(x_u + y_i)$$

Les autres contraintes qui pèsent sur les tâches correspondent au fait qu'elles sont faisables dans le mois. Pour chaque machine il faut que la somme des durées des tâches acceptées sur cette machine ne dépasse pas M. On a donc les deux contraintes :

$$\sum_{i=1}^{n} a_i x_i \leq M, \quad \sum_{i=1}^{n} b_i y_i \leq M$$

D'où le programme mathématique suivant :

$$\begin{cases} Max & \sum_{i=1}^{n} w_i(x_u + y_i) & \\ & x_i + y_i \leq 1 & \forall i \in \{1, \ldots, n\} \\ & \sum_{i=1}^{n} a_i x_i \leq M & \\ & \sum_{i=1}^{n} b_i y_i \leq M & \\ & x_i, y_i \in \{0, 1\} & \forall i \in \{1, \ldots, n\} \end{cases}$$

On définit $F_k(A, B)$ le profit maximal d'un ordonnancement des tâches k à n pour lequel la durée des tâches sur la machine 1 est inférieure ou égale à A, celle sur la machine 2 est inférieure ou égale à B.

2 Décrire le schéma de programmation dynamique induit par cette définition (phases, états, décisions, transitions, etc)

Les phases sont les différentes tâches, l'état (A, B) est la quantité résiduelle de temps sur chacune des machines, à la phase k dans l'état (A, B) les décisions possibles sont :

- δ_k^0 : on rejette la tâche k. On aboutit dans l'état (A, B) avec un profit immédiat nul.
- δ_k^1 : on accepte la tâche k sur la machine 1 (c'est possible seulement si $a_k \leq A$). On aboutit à l'état $(A - a_k, B)$ avec un profit w_k.
- δ_k^2 : on accepte la tâche k sur la machine 2 (c'est possible seulement si $b_k \leq B$). On aboutit à l'état $(A, B - b_k)$ avec un profit w_k.

3 Donner la valeur de $F_n(A, B)$ en fonction des données du problème et de A, B

On se pose la question ici pour une tâche (la dernière). On a donc $F_n(A, B) = w_n$ si la tâche n peut être mise sur l'une des machines (c'est à dire si $a_n \leq A$ ou $b_n \leq B$) et $F_n(A, B) = 0$ sinon.

4 Indiquez l'équation de récurrence satisfaite par $F_k(A, B)$.

Il suffit d'appliquer ce que vous avez vu en cours du schéma général de programmation dynamique :

$$F_k(A, B) = \begin{cases} \max(F_{k+1}(A, B), w_k + F_{k+1}(A - a_k, B), w_k + F_{k+1}(A, B - b_k)) & si \quad a_k \leq A, b_k \leq B \\ \max(F_{k+1}(A, B), w_k + F_{k+1}(A - a_k, B)) & si \quad a_k \leq A, b_k > B \\ \max(F_{k+1}(A, B), w_k + F_{k+1}(A, B - b_k)) & si \quad a_k > A, b_k \leq B \\ F_{k+1}(A, B) & si \quad a_k > A, b_k > B \end{cases}$$

5 Ecrire l'algorithme qui découle de ces égalités permettant de trouver un ordonnancement optimal. On précisera sa complexité.

L'algorithme consiste tout d'abord à remplir un tableau T à 3 dimensions. Le premier axe varie de 1 à n et correspond aux tâches, le second et le troisième correspondent aux états de 0 à M. Il s'agit de remplir progressivement le tableau de sorte que $T[k, A, B] = F_k(A, B)$.

Phase d'initialisation : complexité $O(M^2)$

Pour A de 0 à M
  Pour B de 0 à M
  si $a_n \leq A$ ou $b_n \leq B$ alors $T[n, A, B] = w_n$
  sinon $T[n, A, B] = 0$

Phase de calcul : complexité $O(nM^2)$

Pour k de $n-1$ à 1
  Pour A de 0 à M
    Pour B de 0 à M
    $T[k, A, B] = T[k+1, A, B]$
    si $a_k \leq A$ alors $T[k, A, B] = \max(T[k, A, B], w_k + T[k+1, A - a_k, B])$
    si $b_k \leq B$ alors $T[k, A, B] = \max(T[k, A, B], w_k + T[k+1, A, B - b_k])$

Génération de la solution optimale : complexité $O(n)$

$A = M$, $B = M$
Pour k de 1 à $n-1$
  si $T[k, A, B] = T[k+1, A, B]$ alors $x_k = y_k = 0$
  si $a_k \leq A$ et $T[k, A, B] = w_k + T[k+1, A - a_k, B]$ alors $x_k = 1, y_k = 0, A = A - a_k$
  sinon $x_k = 0, y_k = 1, B = A - b_k$
si $T[n, A, B] = 0$ alors $x_k = y_k = 0$
sinon si $a_n \leq A$ alors $x_k = 1, y_k = 0$ sinon $x_k = 0, y_k = 1$

Exercice 3 Algorithmes approchés (3 points)

Considérons un problème de minimisation sous la forme :

$$\begin{cases} \min f(X) \\ X \in D \end{cases}$$

On dispose d'un algorithme $\mathcal{A}$ qui résoud ce problème en fournissant une solution réalisable mais pas toujours optimale.

1 Que signifie l'assertion suivante : "'l'algorithme $\mathcal{A}$ est 2-approché ?"'

Si $X^{\mathcal{A}}$ désigne la solution calculée par l'algorithme, et si X^{opt} désigne la solution optimale on dit que l'algorithme eest 2-approché lorsque pour toute instance du problème on a toujours :

$$\frac{f(X^{\mathcal{A}})}{f(X^{opt})} \leq 2$$

2 Indiquez un algorithme vu en cours qui a cette propriété.

First Fit pour le problème du bin packing.