# TP : Problème du sac-à-dos et Branch-and-Bound.

Un voleur s'introduit dans le manoir d'un riche lord anglais. Il veut s'emparer d'un butin de la plus grande valeur possible, mais doit le transporter dans un sac-à-dos dont la capacité est limitée à une masse totale $W$. Il fait rapidement une liste des objets dont il peut s'emparer, avec leurs poids et une estimation de leur valeur. Quels objets doit-il alors choisir ?

Mathématiquement, notons $I = \{1, \ldots, n\}$ l'ensemble des objets disponibles, et $w_i, v_i$ respectivement le poids et la valeur de l'objet $i$, pour tout $i \in I$. Une solution du problème est un sous-ensemble $J \subseteq I$ (d'objets pris) tel que $\sum_{i \in J} w_i \leq W$. L'objectif est de maximiser la valeur des objets pris $\sum_{i \in J} v_i$.

Ce problème est NP-difficile : on n'espère donc pas qu'il existe un algorithme polynomial exact pour le résoudre. Nous utilisons donc un algorithme qui est peut-être exponentiel, mais qui en pratique se révèle souvent très efficace, le *branch and bound* (littéralement : brancher et borner).

La technique du *branch and bound* est toujours la même et se décompose en plusieurs étapes :

1. Écrire un programme linéaire en nombres entiers avec des variables binaires (on peut en fait adapter la technique pour des variables quelconques, mais c'est plus simple ainsi). On écrit donc un programme linéaire, mais on contraint les variables à prendre une valeur 0 ou 1 dans la solution. Pour nous, posons pour tout objet $i \in I$ la variable $x_i$, valant 1 si on prend l'objet et 0 sinon. Nous pouvons alors poser, en utilisant les équations ci-dessus, le programme linéaire en nombres entiers suivant :

$$\begin{aligned} &\max \sum_{i \in I} v_i x_i \\ \text{sous contrainte } &\sum_{i \in I} w_i x_i \leq W \qquad\qquad x_i \in \{0,1\} \qquad (\forall i \in I) \end{aligned}$$

2. Avec ce programme linéaire nous pouvons facilement trouver une solution fractionnaire (qui reviendrait à couper des objets en morceaux). Cette solution est prise dans un ensemble plus grand de solutions admissibles (pas seulement 0 ou 1), elle est donc au moins aussi bonne que la meilleure solution entière ; elle fournit donc une *borne supérieure* sur la valeur optimale d'un sac-à-dos entier.

   De plus, même si on force certains objets à être ou ne pas être dans le sac-à-dos, on peut aussi donner une borne supérieure : on retire les variables de ces objets du programme linéaire, on ajuste la contrainte de masse et la valeur de l'objectif en fonction des objets dans le sac-à-dos.

   Le second ingrédient est donc de pouvoir donner une borne supérieure de l'objectif, lorsque certaines variables sont forcées à 0 ou à 1. C'est toujours possible en remplaçant dans le programme linéaire, les variables forcées par leur valeur, et en résolvant la relaxation fractionnaire du programme linéaire obtenu.

3. (Branch) On construit ensuite un arbre de décision binaire, de hauteur $|I|$, et à chaque feuille correspond une affectation 0,1 des variables.

   Pour cela, indiçons par 0 les arêtes entre un nœud et son fils gauche et par 1 les arêtes entre un nœud et son fils droit. Alors la suite des indices $x_1, x_2, x_3, \ldots, x_n$ entre la racine et une feuille quelconque de l'arbre correspond à une affectation des variables.

   De plus, à tout nœud de profondeur $l$ correspond une affectation des variables $x_1, x_2, \ldots, x_l$, par la même méthode. Descendre dans cet arbre correspond donc à raffiner des solutions partiellement entières, avec de moins en moins de variables non-entières.

Bien sûr, cet arbre contient un nombre exponentiellement de nœuds, nous n'allons donc pas le construire en entier. Mais notre problème est de trouver la feuille dont la solution correspondante est admissible et de valeur maximale, en explorant l'arbre.

4. (Bound) Pour être efficace, il nous faut élaguer l'arbre en supprimant des branches. Mais nous ne voulons pas supprimer la solution optimale. Nous avons deux façons de nous assurer qu'un nœud ne possède pas comme descendant de feuille qui soit une solution optimale :
   — si le nœud ne possède pas de solution partielle réalisable, dans notre cas parce que l'ensemble des objets dont la variable est déjà fixé à 1 dépasse la capacité du sac-à-dos, alors le sous-arbre de ce nœud ne possède pas de solutions réalisables. Ce n'est donc pas utile de l'explorer.
   — si la borne supérieure de ce nœud est inférieure à la valeur d'une solution que nous avons déjà trouvée, alors toutes les feuilles admissibles ont une valeur moindre que celle que nous connaissons déjà, et ne sont donc pas optimales. De nouveau, il n'est pas nécessaire de parcourir le sous-arbre de ce nœud.

   Si les bornes supérieures que nous calculons sont de bonne qualité, on peut espérer supprimer une très grand nombre de sous-arbres et ainsi pouvoir rapidement trouver la solution optimale.

Votre travail consiste à utiliser cette méthode pour résoudre le problème du sac-à-dos. Vous utiliserez le langage de programmation de votre choix pour cela [1]. Il n'est pas nécessaire d'utiliser GLPK ou des solveurs de programmes linéaires pour trouver les bornes supérieures : un algorithme glouton permet de calculer la solution fractionnaire optimale (prendre successivement l'objet disponible de meilleur ratio $\frac{\text{valeur}}{\text{masse}}$, quitte à le couper s'il ne rentre pas).

Vous ordonnerez les objets par ratio décroissant, et explorez l'arbre en allant prioritairement à gauche (prendre l'objet). Ceci permettra à l'algorithme de trouver rapidement une solution de bonne qualité, et donc entrainera un meilleur élagage de l'arbre.

Nous vous fournissons des fichiers contenant les données sous la forme suivante : un objet par ligne, chaque ligne étant un flottant (le poids) suivi d'un autre flottant (la valeur de l'objet), à l'exception de la première ligne qui contient un seul flottant : la capacité du sac-à-dos.

Vous pouvez aussi écrire un fichier pour `glpsol`, celui-ci sait faire des *branch and bound* automatiquement, mais utilisez-le uniquement pour vérifier que vos solutions sont correctes.

Les instances à résoudre sont disponibles sur cette page :
`http://pageperso.lif.univ-mrs.fr/~basile.couetoux/enseignement.html`

Ce devoir devra être réalisé pendant les TP 2 et 3. Il sera noté. **Nous tiendrons compte de la qualité d'écriture et de la qualité algorithmique de votre programme dans notre notation**. Comme toujours, si vous utilisez du matériel (code source) qui ne vous appartient pas, vous le signalez dans un fichier `README`. Vous rendrez votre projet à l'issue du TP 3, sous forme d'un unique fichier archive (format `zip` ou `tar.gz`) contenant vos sources, et dont le nom contiendra vos patronymes. Vous pouvez rendre seul ou en binôme.

1. C'est bien plus facile avec un langage fonctionnel