Chapitre 17 : Complément d'algorithmique

Table des matières

1 Optimisation

1.1 Optimisation exacte

1.1.1 Introduction

On s'intéresse ici à la résolution de problèmes d'optimisation au sens de la définition du chapitre 16, en 1.2.2 : on cherche un algorithme calculant une solution optimale **pour toute instance**.

1.1.2 Exemple : le problème du sac à dos

Le problème est le suivant : on dispose d'objets de poids respectifs $w_0, \dots, w_{n-1}$ et de valeurs respectives $p_0, \dots, p_{n-1}$ et d'un sac à dos capable de supporter un poids W. On souhaite sélectionner des objets de sorte à maximiser la valeur totale sans dépasser la capacité du sac à dos.

Dans la version en variables réelles, on suppose que l'on peut prendre des fractions des objets. Le problème d'optimisation se formule ainsi :

Maximiser $\displaystyle\sum_{i=0}^{n-1} x_i p_i$ sous les contraintes :

$$\begin{cases} \displaystyle\sum_{i=0}^{n-1} x_i w_i \leqslant W \\ \forall i \in [\![0\ ;\ n-1]\!]\,,\ x_i \in [0\ ;\ 1] \end{cases}$$

Ce problème est résolu par un algorithme glouton :

Algorithm 1: Solution du problème du sac à dos, version en variables réelles

```
Trier les objets par p_i/w_i décroissant;
while que possible en considérant les objets dans cet ordre do
    Fixer x_i à 1;
Lorsque cela n'est plus possible, prendre la fraction de l'objet courant
  permettant de remplir le sac;
```

Cet algorithme calcule bien une solution optimale : si on note i l'objet de $\frac{p_i}{w_i}$ maximal non encore sélectionné et si une solution optimale coïncidant avec l'algorithme sur les objets avant i, et ne sélectionne pas cet objet dans son intégralité, alors $\exists j$ tel que la solution optimale sélectionne une fraction de l'objet j qui est $> x_j$.

Dans ce cas, il existe $\delta_j > 0$ tel que l'on peut ajouter une quantité $\frac{\delta_j}{w_i}$ de l'objet i et retirer une quantité $\frac{\delta_j}{w_j}$ de l'objet j à la solution optimale.

La variation de poids est $\frac{\delta_j}{w_i} w_i - \frac{\delta_j}{w_j} w_j = 0$, donc on a toujours une solution.

La variation de valeur est

$$\frac{\delta_j}{w_i}p_i - \frac{\delta_j}{w_j}p_j = \underbrace{\delta_j}_{>0}\underbrace{\left(\frac{p_i}{w_i} - \frac{p_j}{w_j}\right)}_{\geqslant 0}$$

Donc la solution reste optimale.

On peut donc modifier la solution optimale jusqu'à l'obtention d'une solution optimale ayant choisi l'objet i dans son intégralité.

L'invariant « il existe une solution optimale ayant fait les mêmes choix que l'algorithme glouton » est vrai.

Remarque : le problème du sac à dos, dans sa version entière (les $x_i \in \{0, 1\}$) ne peut pas être résolu par l'algorithme glouton (algorithme n°1) auquel on retire la dernière étape ne prenant qu'une fraction du dernier objet.

Poids	5	5	7
Valeur	5	5	8

$W = 10$

Cf l'exemple ci-dessus : l'algorithme glouton donne une solution de valeur 8 en prenant l'objet de poids 7 alors qu'une solution optimale est de valeur 10 : on prend les deux objets de poids 5.

1.1.3 Séparation et évaluation (*branch and bound*)

• Une technique de résolution des problèmes d'optimisation consiste à effectuer une exploration exhaustive de l'ensemble des solutions et à conserver la meilleure solution.

Cependant, on se heurte à des problèmes de complexité (exemple : pour le sac à dos en variables entières, il y a 2^n solutions potentielles à tester).

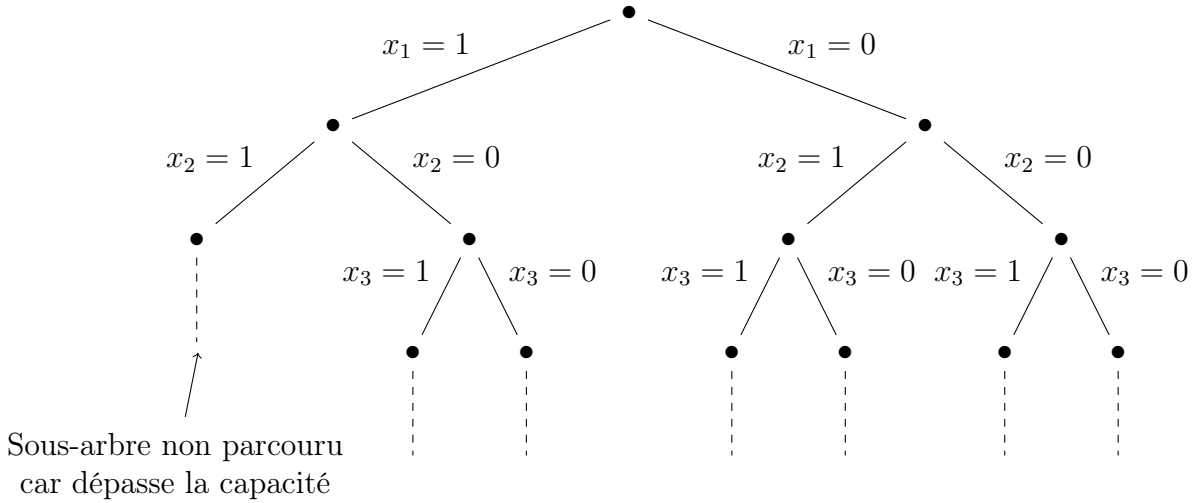
On peut parfois accélérer la recherche grâce à l'heuristique du retour sur trace (*cf* chapitre 8, 4.3).

Par exemple, pour le problème du sac à dos, il y a surement de nombreuses combinaisons d'objets qui dépassent la capacité du sac à dos. On peut donc sélectionner les objets un à un et lorsque l'on s'aperçoit que la capacité du sac à dos est dépassée, on revient sur le dernier choix.

En pratique, cela revient à construire un arbre binaire dans lequel tous les nœuds de même profondeur correspondent à un même objet et pour ces nœuds, le fils gauche correspond au cas où l'on a sélectionné l'objet et le fils droit au cas où l'objet n'est pas sélectionné.

On élague les branches correspondant à des sélections dépassant la capacité du sac à dos.

Si $w_1 + w_2 > W$,



Dans le cadre de la résolution d'un problème d'optimisation, on peut parfois élaguer encore plus l'arbre de recherche en considérant le coût des solutions construites : si on sait évaluer une borne du meilleur possible pour une série de choix sans parcourir l'intégralité du sous-arbre correspondant, on peut parfois élaguer ce sous-arbre si on connaît déjà une solution de coût meilleur que cette borne.

Cette méthode consiste à concevoir un algorithme par séparation et évaluation :

– La séparation consiste à diviser le problème en sous-problèmes, donc à créer un branchement dans l'arbre de reverche.

Exemple : pour le problème du sac à dos, on a deux sous-problèmes, selon que l'objet i est sélectionné ou non.

– L'évaluation consiste à déterminer une borne sur le coût d'une solution optimale **réalisable avec les choix déjà faits** et à le comparer avec une borne connue pour savoir s'il est nécessaire de poursuivre l'exploration du sous-arbre.

Pour que cette méthode soit efficace, on a besoin de bonnes heuristiques pour :

– La séparation : si les choix initiaux convergent rapidement vers une « bonne solution », on élaguera plus de branches dans la suite de l'exploration.

– L'évaluation : on doit pouvoir calculer *efficacement* une borne *la plus juste possible* pour avoir de bonnes chances d'élaguer des branches.

1.1.4 Exemple : le problème du sac à dos

• Pour la séparation, on sélectionne ou pas un objet, avec l'heuristique suivante : on considère les objets par $\frac{p_i}{w_i}$ décroissant.

• Pour l'évaluation, on utilise l'heuristique de relaxation : on relâche certaines contraintes, ce qui élargit le domaine des solutions donc permet potentiellement d'atteindre un meilleur coût. Si le problème relâché est plus simple à résoudre, le coût d'une solution optimale est donc la borne recherchée.

Ici, on effectue une relaxation continue : on n'impose plus aux x_i d'être des entiers,

ce qui nous ramène au problème vu en 1.1.2 (page 2), que l'on sait résoudre efficacement (en $\mathcal{O}(n)$ car les objets seront triés une seule fois au début de l'algorithme pour l'heuristique de séparation).

Remarque : si l'algorithme nous donne une solution entière : on a trouvé la solution optimale (selon les choix qui sont déjà fait).

Si l'algorithme nous donne une solution avec un terme dans $]0, 1[$, la partie entière de la valeur de cette solution est une borne supérieure sur la solution optimale du problème entier.

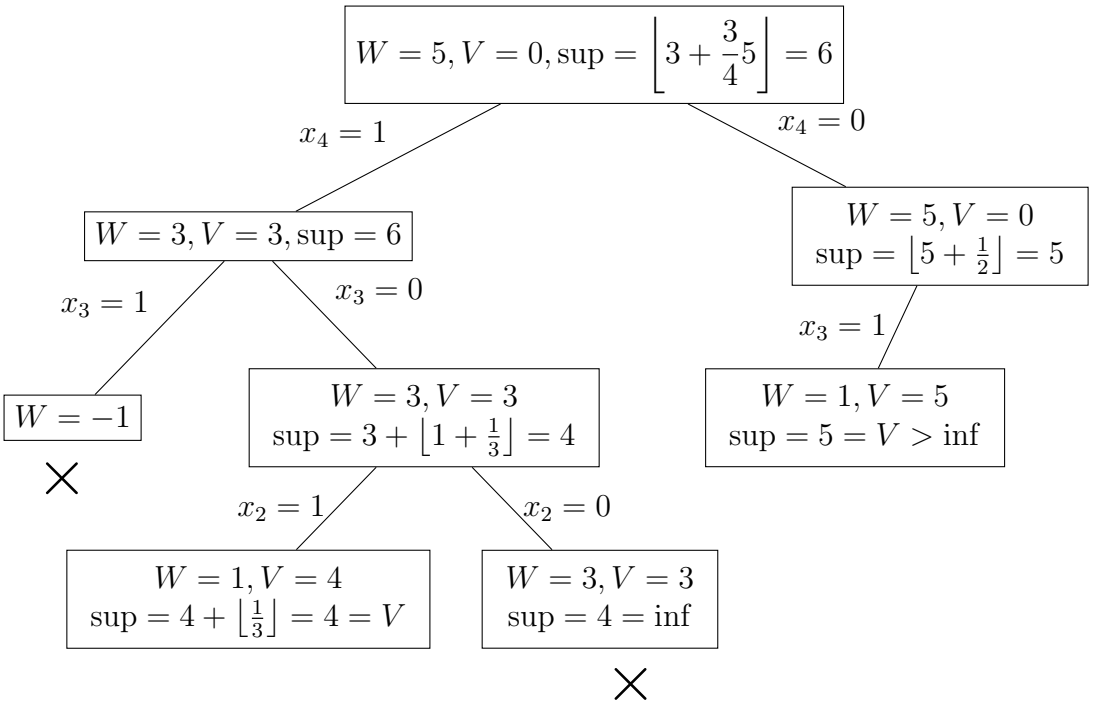
Exemple : on considère l'instance suivante :

i	1	2	3	4
p_i	1	1	5	3
w_i	3	2	4	2

$W = 5$

Le tri des objets indique qu'on doit les traiter dans l'ordre suivant : 4, 3, 2, 1.

On note au cours de l'exécution, W la capacité courante du sac à dos, et V la valeur totale courante des objets sélectionnés.



Solution $(0, 1, 0, 1)$, et $\inf = 4$

Solution $(0, 0, 1, 0)$, et $\inf = 5$

Conclusion : sélectionner uniquement l'objet 3 est une solution optimale.

Exo écrire l'exécution de l'algorithme si l'heuristique de séparation consiste à prendre les objets par ordre d'indice ou par ordre de poids décroissant ou de valeur décroissante.

Tester aussi l'heuristique d'évaluation qui consiste à prendre la borne des valeurs des objets comme borne supérieure.