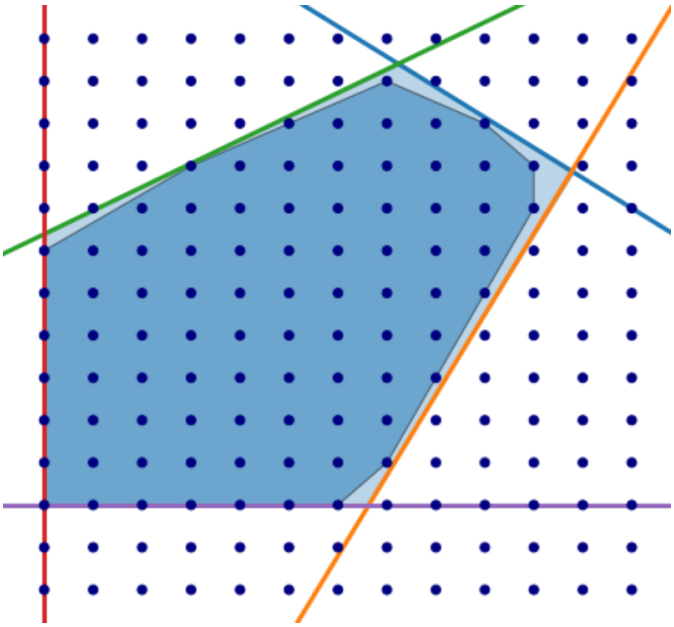
Université Paris-Saclay

Technique d'Optimisation de la Parallélisation

Optimisation et Recherche Opérationnelle

Multiple 0/1 Knapsack problem



LOZES Benjamin

9 mai 2022

université PARIS-SACLAY

Table des matières

0 Introduction

L'objectif de ce projet est de programmer un algorithme de résolution par *Séparation et évaluation* (plus connu sous l'appelation anglais *Branch and Bound*), sur le problème de notre choix.

J'ai pour ma part choisi d'implémenter un problème de sac-à-dos unidimensionnel, puis multi-dimensionnel. Les deux solutions sont implémentées de façon distincte car je n'avais pas prévu initialement la version multi-dimensionelle.

L'implémentation est disponible sur mon github, avec les instructions nécessaires à son utilisation.

1 0/1 Knapsack problem

Définition du problème : Le problème du sac-à-dos est un problème d'optimisation combinatoire, définit comme suit : "Étant donné n objets, ayant chacun une valeur et un poids associés, remplir un sac-à-dos de capacité C avec ces objets, de manière à ce qu'il contienne la plus grande valeur cumulée possible, sans dépasser sa capacité maximale." Ce problème nécessite également une résolution en nombres entiers, puisqu'il est interdit de fractionner un objet, autrement dit, un objet est soit choisi, soit laissé.

Le problème du sac-à-dos multi-dimensionnel reprend exactement les mêmes règles, sauf que nous disposons de m sac-à-dos, ayant tous une capacité limite identique C. Je précise que l'optimisation ne consistera pas à répartir les objets de manière à avoir le moins de poids par sac-à-dos, cela serait plutôt un problème d'*ordonnancement sur machines identiques*, également résolvable par méthode de *séparation et évaluation*.

2 Méthode de séparation et évaluation

Définition de l'algorithme : La méthode de séparation et évaluation vise à résoudre des problèmes d'optimisation combinatoire, tel que le problème du sa-à-dos. Il s'agit plus concrètement de trouver une valeur, dit optimale, minimisant un *coût* définit dans le problème par une fonction, le tout en contraignant ce coût sous une valeur barrière.
Cette méthode s'appuie sur deux phases distinctes :

— Séparation : Phase consistant à diiser le problème en sous-problèmes ayant chacun potentiellement plusieurs soltions réalisables. Cette subdivision, souvent visualisée comme un arbre de recherche/décision, permet de trouver la solution optimale au problème d'origine. Néanmoins, si l'on s'arrête à cette étape il ne s'agit que d'un algorithme de *bruteforce*, consistant à énumérer toutes les solutions possibles.

— Évaluation : Cette phase permet de déterminer puis de comparer la meilleure valeur trouvée lors du parcours des sous-problèmes. En d'autres termes, il s'agit d'estimer par une valeur palier, si le sous-problème courant pourrait in fine être meilleur que ceux trouvés précedemment. Cela s'implémente avec différentes méthodes de relaxations. De plus, une seconde valeur palier est mise en place, elle consiste à restreindre le problème à une contrainte globale (ici la capacité du sac-à-dos).

3 Version uni-dimensionnelle

L'implémentation de cette version est réalisée en C, les structures sous-jacentes ont été définies spécifiquement pour ce projet. Néanmoins, le code présenté n'est ni parallélisé, ni profondément optimisé, l'objectif étant avant tout l'implémentation à titre expérimental.

3.1 Structure mémoire

Toutes les structures sont implementées avec un selon une norme SoA (Structure of Array), cela afin d'optimiser les accès mémoires.

Le problème est implémenté comme suit :
- Liste de poids
- Liste de valeurs
- Capacité maximale du sac-à-dos

L'arbre de recherche est implemenété comme étant une file d'attente (FIFO), comme suit :
- Liste de noeuds (norme AoS ici, par soucis de relecture)
- Capacité de la liste de noeuds
- Indice du premier noeud ajouté
- Indice du dernier noeud ajouté

Un noeud de l'arbre de recherche est implementé comme suit :
- Liste de status (SÉLECTIONNÉ / NON-SÉLECTIONNÉ)
- Indice de l'objet courant
- Profondeur du noeud (utilisé à titre informatif seulement)

3.2 Complexité temporelle & spatiale

Cette méthode, bien que plus efficace, dans la plupart des cas, qu'une méthode *bruteforce*, possède néanmoins un pire cas revenant à énumérer tous les sous-problèmes possibles (comme un *bruteforce*).

Sa complexité temporelle est donc de $O(n^2)$ dans le pire cas, et de $O(n)$ dans le meilleur.

Cette méthode reposant sur le parcours d'un arbre de recherche, sa complexité spatiale est exponentielle.

3.3 Calcul de la borne supérieure

Cette borne supérieure correspond à la somme des valeurs des objets sélectionné dans le noeud courant, **avec** l'autorisation de valeur fractionnaire pour le dernier élement à compter.

Soit X l'ensemble, au noeud n, des objets sélectionnés, et C la capacité du sac-à-dos. La borne supérieure est définit par :
$V_e = \sum_i^X X_i \leq C$
$B_{sup} = V_e + X_{i+1} * \frac{C}{V_e}$

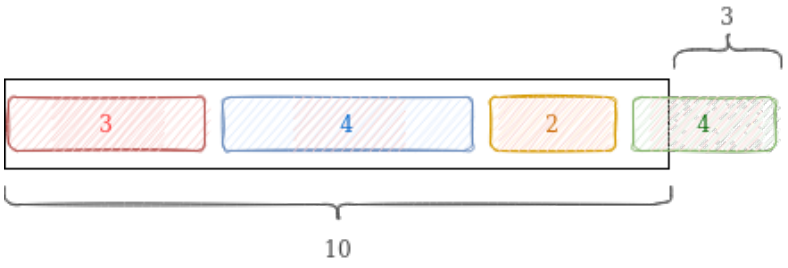


FIGURE 1 – Schématisation : Borne supérieure (égale à 10)

3.4 Calcul de la borne inférieure

Cette borne inférieure correspond à la somme des valeurs des objets sélectionné dans le noeud courant, **sans** l'autorisation de valeur fractionnaire pour le dernier élement à compter.

Soit X l'ensemble, au noeud n, des objets sélectionnés, et C la capacité du sac-à-dos. La borne inférieure est définit par :
$B_{inf} = \sum_i^X X_i$ si $B_{inf} \leq C$
$B_{inf} =$ NON-RÉALISABLE sinon

Néanmoins, cette borne a surtout pour fonction de vérifier la faisabilité d'une solution, n'acceptant pas de partie fractionnaire, tous les objets sélectionnés doivent pouvoir rentrer dans le sac, sans quoi la solution n'est pas réalisable.

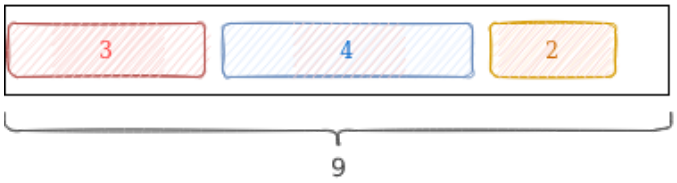


FIGURE 2 – Schématisation : Borne inférieure

3.5 Développement en sous-problèmes

La développement de l'arbre des sous-problèmes se fait selon deux actions possibles, prendre ou laisser un objet (statut SÉLECTIONNÉ / NON-SÉLECTIONNÉ). Il s'agit là d'actions spécifiques au problème du sac-à-dos, dans un problème de *Voyageur de commerce* nous aurions à la place choisis une ville où aller parmis toute celles non-visitées. Nous verrons plus tard que dans le cadre du sac-à-dos multi-dimensionnel, ces actions sont plus générales.

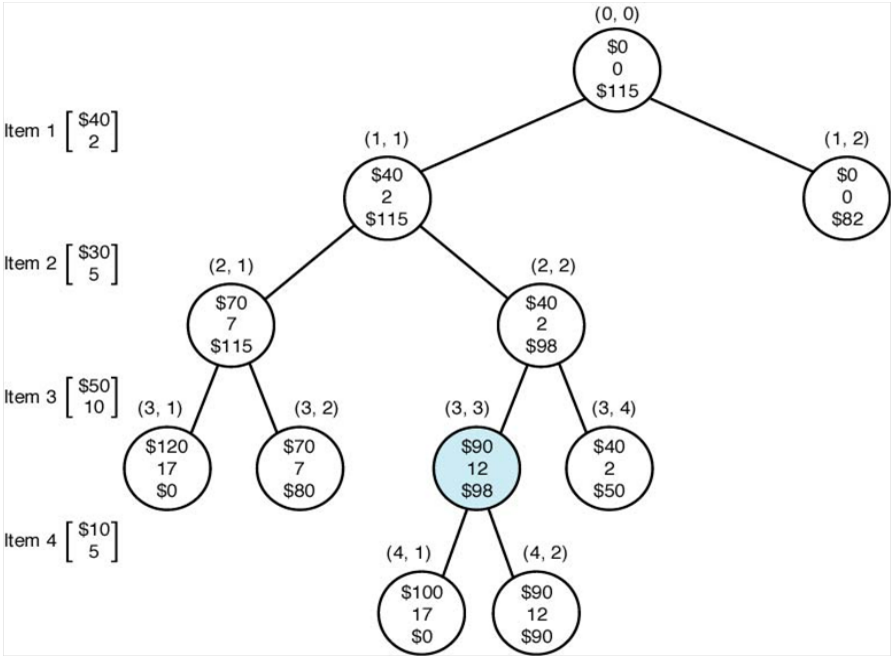


FIGURE 3 – Exemple : Arbre de décision sur un problème de sac-à-dos

4 Version multi-dimensionnelle

L'implémentation de cette version est également réalisée en C, les structures sous-jacentes reprennent celles du sac-à-dos unidimensionnel. Notons que l'algorithme d'une sac-à-dos multi-dimensionnel inclus dans ses possibilités le problème du sac-à-dos unidimensionnel (avec un ssac-à-dos à 1 dimension).
Ces deux algorithmes sont séparés pour des raisons pratiques, j'ai en effet commencé par l'implémentation de la version unidimensionnelle, et ai préféré la garder à part pour pouvoir l'expliquer plus clairement.

4.1 Structure mémoire

Seule la structure des noeuds à étée modifiée pour cet algorithme, les autres ont été rappelées ci-dessous par soucis de facilité de lecture.

Le problème est implémenté comme suit :
- Liste de poids
- Liste de valeurs
- Capacité maximale du sac-à-dos

L'arbre de recherche est implementé comme étant une file d'attente (FIFO), comme suit :
- Liste de noeuds (norme AoS ici, par soucis de relecture)
- Capacité de la liste de noeuds
- Indice du premier noeud ajouté
- Indice du dernier noeud ajouté

Un noeud de l'arbre de recherche est implementé comme suit :
- Liste de status (SÉLECTIONNÉ / NON-SÉLECTIONNÉ)
- **List des assignations (problème multi-dimensionnel uniquement)**
- Indice de l'objet courant
- Profondeur du noeud (utilisé à titre informatif seulement)

nb : la liste d'assignation n'est absolument pas nécessaire dans le problème multi-dimensionnel originel, néanmoins, dans un soucis de compatiblité des structures entre les deux algorithmes j'ai préféré rajouter cette liste.

4.2 Complexité temporelle & spatiale

La complexité de cet algorithme reste identique à celle de la version unidimensionnelle.

4.3 Calcul de la borne supérieure

Cette borne supérieure correspond à la somme **maximal** des valeurs des objets sélectionné dans le noeud courant, **parmis les sac-à-dos**, **avec** l'autorisation de valeur fractionnaire pour le dernier élement à compter.

Soit X l'ensemble, au noeud n, des objets sélectionnés et assignés, et C la capacité des sac-à-dos. La borne supérieure est définit par :
$V_{es} = \sum_i^X X_i \leq C$
$B_{sup_s} = V_e + X_{i+1} * \frac{C}{V_{es}}$
$B_{sup} = MAX(B_{sup_0}, B_{sup_1}, ...)$

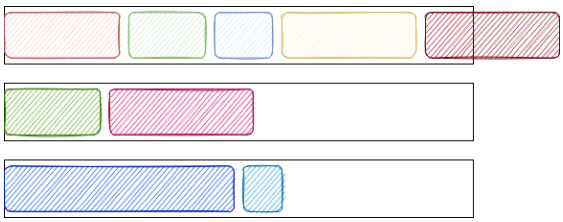


FIGURE 4 – Schématisation : Borne supérieure sur sac-à-dos multi-dimensionnel

4.4 Calcul de la borne inférieure

Cette borne inférieure correspond à la somme des valeurs des objets sélectionné dans le noeud courant, **sans** l'autorisation de valeur fractionnaire pour le dernier élement à compter.

Soit X l'ensemble, au noeud n, des objets sélectionnés et assignés, et C la capacité des sac-à-dos. La borne inférieure est définit par :
$B_{inf_s} = \sum_i^X X_i$
$B_{inf} = MAX(B_{inf_0}, B_{inf_1}, ...)$ si $\forall s, B_{inf_s} \leq C$
$B_{inf_s} =$ NON-RÉALISABLE sinon

Néanmoins, cette borne a surtout pour fonction de vérifier la faisabilité d'une solution, n'acceptant pas de partie fractionnaire, tous les objets sélectionnés doivent pouvoir rentrer dans le sac **dans lequel ils sont assignés**, sans quoi la solution n'est pas réalisable.

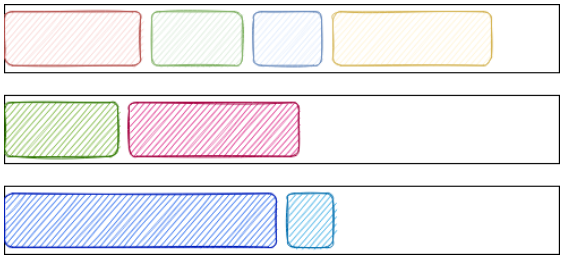


FIGURE 5 – Schématisation : Borne inférieure sur sac-à-dos multi-dimensionnel

4.5 Développement en sous-problèmes

La développement de l'arbre des sous-problèmes se fait selon $S + 1$ actions possibles, S étant le nombre de dimensions, nous pouvons placer l'objet dans l'une des dimension, ou bien ne le placer nulle part. En cela, ce problème se rapproche plus d'une problème de Voyageur de commerce dans sa résolution.
Cette fois-ci, notre problème ne se décompose donc pas en arbre binaire comme sa version unidimensionnelle, mais en un arbre d'autant plus large qu'il admet de dimensions. Son temps de résolution est donc exponentiellement plus coûteuse en espace, et en temps de calcul.

```
Slot 00: | 00 | 02 | -- |  --> {weight: 55  value: 20}
Slot 01: | 01 | 05 | 08 |  --> {weight: 57  value: 30}
Slot 02: | 03 | 04 | -- |  --> {weight: 57  value: 30}
Slot 03: | 06 | 07 | -- |  --> {weight: 60  value: 35}
```

FIGURE 6 – Solution pour le problème donné en exemple sur le dépît Github