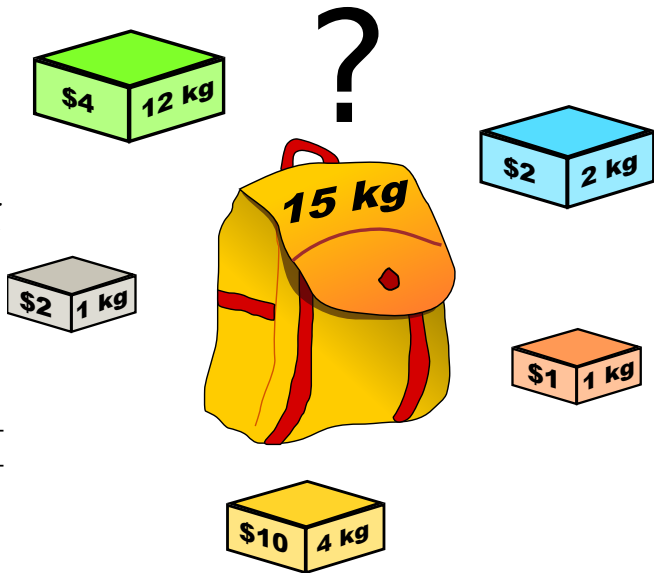
Problème du sac à dos

Chap. 23,02

1 Introduction

Dans ce TD, on s'intéresse à une classe de problèmes d'optimisation connus sous le nom général de « problème du sac à dos ». On peut définir ce problème de la manière suivante : « *durant un cambriolage un voleur possède un sac dont la capacité (en poids par exemple) est limitée. Il se trouve face à un ensemble d'objets qu'il veut dérober. Chacun de ces objets est caractérisé par sa valeur et son poids. Le voleur souhaite optimiser la valeur totale des objets qu'il dérobe tout en ne dépassant pas le poids maximal supporté par son sac* ».

Ce problème est une abstraction pour un grand nombre d'autres problèmes d'optimisation. Il a été utilisé en cryptographie comme base pour différents schémas de chiffrement[1], il est utilisé lors du chargement des bateaux ou d'avions, lors de la découpe de matériaux (minimisation des coupes « chutes » lors de la découpe), etc.



https://fr.wikipedia.org/wiki/Problème_du_sac_à_dos

2 Problème étudié

On considère un sac à dos de masse maximale $m = 40$ kg dans lequel on souhaite ranger les objets dont les caractéristiques sont données ci-dessous :

Objet	A	B	C	D	E	F
Masse	13	12	8	10	14	18
Valeur	700	500	200	300	600	800

Quels objets faut-il sélectionner de façon à ce que la valeur totale, dans le sac à dos, soit maximale ?

3 Méthode de résolution approchée : stratégie gloutonne

Remarque. Une solution se trouve à cette adresse : https://repl.it/@dlatreyte/sacados

1) Importer les types `List`, `Dict` et `Tuple` du module `typing` :

```
from typing import List, Dict, Tuple
```

2) Dans la fonction `main`, définir la liste `objets` dont les membres sont des dictionnaires représentant les différents objets du problème.
Chacun de ces dictionnaires doit donc posséder les clés `'nom'`, `'masse'`, `'valeur'` associées aux valeurs.

```
► objets = [{'nom': 'A', 'masse': 13, 'valeur': 700},
            {'nom': 'B', 'masse': 12, 'valeur': 500},
            {'nom': 'C', 'masse': 8, 'valeur': 200},
            {'nom': 'D', 'masse': 10, 'valeur': 300},
            {'nom': 'E', 'masse': 14, 'valeur': 600},
            {'nom': 'F', 'masse': 18, 'valeur': 800}
            ]
```

3) Dans la fonction `main`, définir la variable `masse_max` et lui affecter la valeur 40.

```
► masse_max = 40
```

Le principe de la stratégie gloutonne consiste à ajouter en priorité les objets « les plus efficaces ». « Plus efficace » ne signifie pas « plus grande valeur » mais « plus grande valeur comparativement à la masse ».

4) Avant d'appliquer l'algorithme glouton, il est nécessaire de trier la liste `objets`. Définir la fonction suivante :

```
def tri_objets(objets: List[Dict]) -> List[Dict]:
    """
    Tri la liste de dictionnaires selon les rapports valeur/masse.
    """
    def recupere_rapports(objet):
        return objet['valeur'] / objet['masse']

    objets_tries = sorted(objets, key=recupere_rapports)

    return objets_tries
```

et affecter, dans la fonction `main`, son résultat à la variable `objets`.
Afficher à l'écran la liste triée.

```
► objets = tri_objets(objets)
  print("Objets utilisables : {}".format(objets), end='\n')
```

Remarque. Ne pas oublier d'appeler la fonction `main`.

5) Définir la fonction `construction_sac_a_dos` dont la spécification est :

```
def constrution_sac_a_dos(objets_tries: List[Dict], masse_max: float) ->
Tuple[List[str], int, int]:
    """
    Construction du sac à dos à partir d'une stratégie gloultonne.
    HYPOTHÈSE : la liste de dictionnaires est triée par ordre croissant des rapports
    valeurs/masses.

    Retourne un tuple constitué d'une liste contenant les noms des objets sélectionnés,
    de la valeur du sac à dos et de la masse du sac à dos.
```

```
    """

► def construtionsac_a_dos(objets_tries: List[Dict], masse_max: float) ->
  Tuple[List[str], int, int]:
    """
    Construction du sac à dos à partir d'une stratégie gloultonne.
    HYPOTHÈSE : la liste de dictionnaires est triée par ordre croissant des rapports
    valeurs/masses.

    Retourne un tuple constitué d'une liste contenant les noms des objets sélectionnés,
    de la valeur du sac à dos et de la masse du sac à dos.
    """
    masse_sac = 0
    valeur = 0
    liste_objets = []

    i = len(objets_tries) - 1  # Compteur
    while i >= 0:
        objet = objets_tries[i]  # objet est un dictionnaire
        if (masse_sac + objet['masse']) <= masse_max:
            valeur += objet['valeur']
            masse_sac += objet['masse']
            liste_objets.append(objet['nom'])

        i -= 1  # Décrémentation du compteur

    return (liste_objets, valeur, masse_sac)
```

6) Affecter, dans la fonction `main`, le tuple retourné par la fonction `construction_sac_a_dos` aux variables `sac_a_dos`, `valeur` et `masse_sac`.

```
► sac_a_dos, valeur, masse_sac = construtionsac_a_dos(objets, masse_max)
```

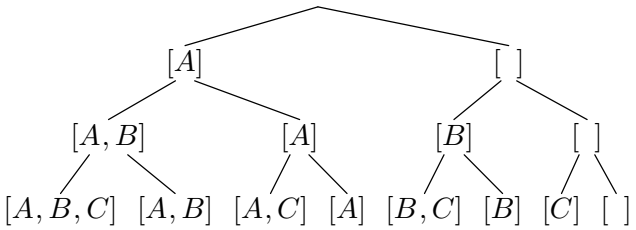
7) Afficher le résultat :

```
print("Stratégie gloutonne")
print("-------------------")
print("Constitution du sac à dos : {}".format(sac_a_dos))
print("Masse du sac à dos : {}".format(masse_sac))
print("Valeur du sac à dos : {}".format(valeur))
print()
```

4 Procédé d'exploration systématique

Si on ne prend pas en compte, dans un premier temps, les contraintes, il est possible d'établir la liste des combinaisons possibles des objets à l'aide d'un arbre d'exploration binaire.

Par exemple, voici ce à quoi cet arbre ressemble lorsqu'on prend en compte les trois premiers objets :

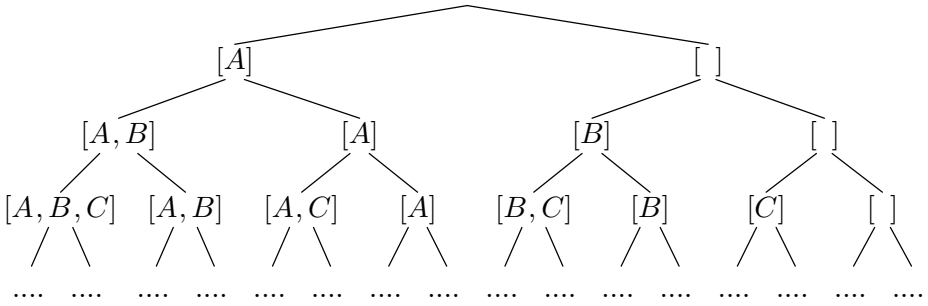


Les combinaisons s'obtiennent en parcourant l'arbre du sommet jusqu'à chaque extrémité : on obtient alors un vecteur dont on peut calculer la poids et la valeur. Il suffit alors de retenir celui dont la masse est inférieure à la masse maximale et dont la valeur est alors maximale.

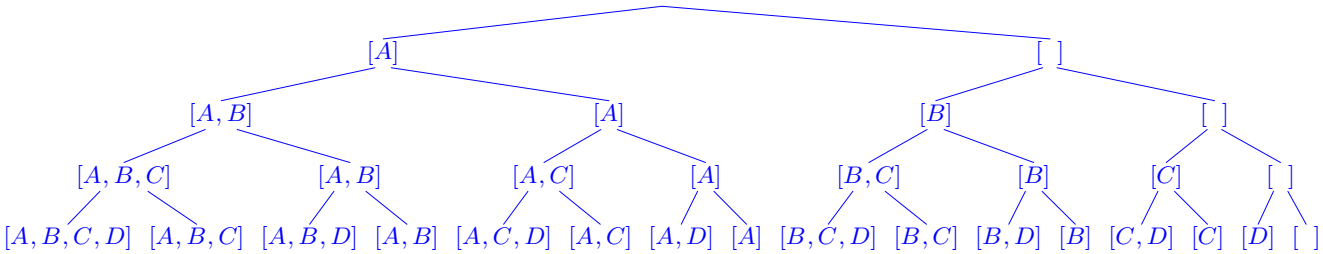
8) Au niveau de chaque nœud à quoi correspond le chemin « gauche » : *objet retenu* ou *objet non retenu* ?

► Objet retenu.

9) Compléter l'arbre avec l'objet D



►



10) Si on implémente un algorithme mettant en œuvre la technique présentée ici, on trouve comme solution $[A, C, F]$. Vérifier que cette solution est meilleure que celle donnée par l'algorithme glouton.
Pourquoi cette méthode n'est cependant pas utilisable la plupart du temps ? Quel est le nombre de combinaisons possibles ?

► Le nombre de combinaisons est 2^n où n est le nombre d'objets. Cette croissance exponentielle rend l'algorithme exact beaucoup trop lent.