1 NSI Activité : un problème de sac à dos

Fichier Prof

Nous possédons 5 poids L=[5 kg, 7 kg, 8 kg, 2kg, 4 kg] et une grande quantité de sacs tous identiques ne pouvant supporter un poids maximal de 10kg.

Combien faudra-t-il de sacs au minimum pour y ranger l'ensemble des 5 objets ?

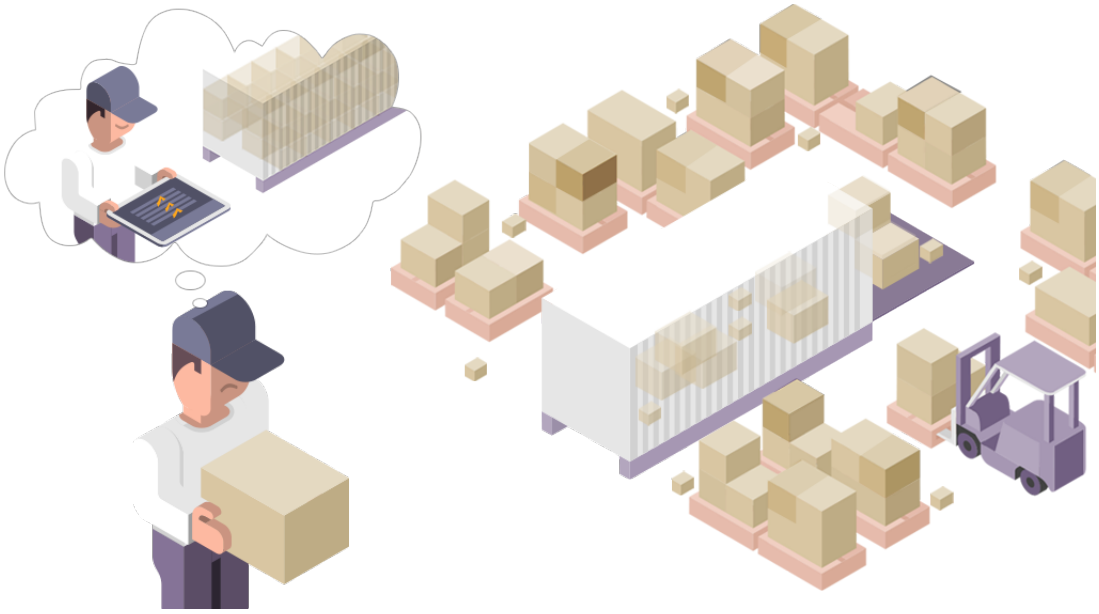


Illustration avec des cartons et des objets à y ranger

1. Que proposez-vous ?

Mise en oeuvre : Vous disposez des 5 verres en carton sur lesquels les poids sont indiqués ainsi qu'un certain nombre d'assiettes identiques représentants les sacs .

Essayez ! Proposez une méthode systématique. Quel est le nombre de sac minimal ? Maximal ?
Cela nous donne une idée du nombre de sacs avec lequel on devrait pouvoir trouver une solution.

2. Par groupe de 2, choisissez un critère et écrire en langage naturel l'algorithme que vous avez choisi.
3. Prenez maintenant deux poids de plus (c'est à dire deux verres de plus) de poids respectifs 5kg et 6 kg. Votre liste de poids est maintenant de taille 7 : L=[5 kg, 7 kg, 8 kg, 2kg, 4 kg, 5 kg, 6 kg].

   Consigne : A deux, un élève prend le matériel en mains, se met dos à l'autre et exécute les commandes données par son camarade.
   Votre algorithme vous fournit-il une solution valide ?

Bilan de l'activité de découverte :

Pour résoudre un problème d'optimisation, on peut faire appel à un algorithme. Celui fournira une solution qui sera dite valide lorsqu'elle solutionne le problème . Cette solution valide sera dite optimale lorsqu'il n'y en a pas de meilleure. Pour prouver qu'une solution n'est pas optimale, il suffit d'en trouver une meilleure !

Il existe plusieurs algorithmes pour répondre au problème posé : les meilleurs sont ceux donnant la réponse optimale qui est 3 sacs ! Parmi ces algorithmes, certains sont caractérisés de gloutons lorsqu'ils fonctionnent de manière itérative en définissant un critère de choix identique à chaque étape semblant sur le moment optimal.

1 NSI Algorithme gloutons

Fichier Prof

1 L'algorithme glouton

Généralisation du problème :

On considère un ensemble de sacs identiques en assez grande quantité, ne pouvant supporter qu'un poids maximal de 20kg. On considère aussi un ensemble de n objets de masses différentes mais connues inférieures ou égales à 10kg :

$$L = [p_1, p_2, p_3, \ldots, p_n]$$

Question : Combien faudra-t-il de sacs au minimum pour y ranger l'ensemble des n objets ?
Le problème a une seule contrainte : le poids de chaque sac ne doit pas dépasser 10 kg.

Notations :

— Chaque objet sera numéroté i et aura donc une masse p_i connue.
— On notera 1 le premier sac, 2 le deuxième etc.

On appellera **solution** du problème la donnée d'une liste

$$S = [[\text{ poids du sac 1}], [\text{ poids du sac 2}], \ldots].$$

Une **solution** est dite **valide** lorsqu'elle répond au problème : parmi celles qui le sont, certaines sont dites **optimales** si elles donnent le nombre de sacs minimum.

C'est un problème d'**optimisation** puisqu'il s'agit de minimiser le nombre de sacs utilisés.

Il existe deux grands types de méthodes de résolution des problèmes d'optimisation
— les méthodes **exactes** c'est à dire permettant d'obtenir une solution optimale à chaque fois.
— les méthodes **heuristiques ou approchées** c'est à dire celles permettant d'obtenir une solution mais non nécessairement optimale.

A retenir .

Un algorithme **glouton** comprend :

1. une construction itérative
2. un critère de choix s'appliquant à l'identique à chaque étape
3. un test d'arrêt

C'est le fait que le choix se fasse de manière « localement optimale » (c'est à dire à chaque étape) qui caractérise la gloutonnerie de l'algorithme.

Dans l'algorithme First Fit Decreasing (FFD) le critère glouton est : on place l'objet le lourd dans le premier sac pouvant le contenir.

a) Déterminer les 3 composantes de l'algorithme FFD.

b) Voici l'algorithme glouton programmé en langage Python : compléter le test.
fichier .py joint et reproduit ici pour l'exposé.

```
def nouvelobjet (t, t2, masse_max) :       # Ajoute l'objet d'indice 0 de la liste t
                                             a un sac de la liste t2
  i =0
  while i<=len(t2)-1 and t2[i]+t[0] > masse_max :
            i += 1
      if i>len(t2)-1 :
            # a completer
      else:
            # a completer
  return (t,t2)

def FFD(t, masse_max) :
      t.sort(reverse=True)
      t2=[]        # liste creee pour recevoir les objets venant de t
      t2.append(t[0])      # le premier element de t2 recoit le premier objet de t
      t.pop(0)             # on supprime de la liste t le premier element qui vient
                                              d etre range dans t2
      while len(t)!= 0 :
            t, t2 = nouvelobjet (t,t2,masse_max)
  return (t, t2)

# t liste des objets
# masse_max : masse maximale que peut contenir un sac"
t=[5,4,8,2,3,2,6]
masse_max = 10
t,t2 = BFD (t, masse_max)
print(len(t2))
```

Vérifiez qu'il vous retourne une solution valide trouvée lors de l'activité de recherche $L = [5,7,8,2,9]$

c) Le tester avec L=[7, 3, 8 , 9, 3, 9] et une masse maximale pour un sac de 20kg : votre solution est-elle optimale ?

d) Nous voulons maintenant tester l'algorithme avec plus de d'objets : $n = 100$. Dans votre programme Python, remplacer t par t=[randint(1,10) for i in range(100)] et une masse maximale pour un sac de 20kg : qu'en concluez-vous ?

Attention, n'oubliez pas d'importer randint de la bibliothèque random !

e) Comparer le nombre de sacs obtenu précédemment au nombre de « sacs de sable ». Qu'en concluez-vous ?

f) A la vue du programme en langage Python, évaluez la complexité de cet algorithme en fonction du nombre n d'objets.

2 Méthode exacte

La question qui se pose est de savoir s'il et « envisageable » de cherche une solution optimale. On se propose donc d'explorer toutes les configurations possibles en commence par le nombre minimal de sac (sac de sable).

Dans la suite la capacité des sac est fixée à 20 et le poids des objets ne dépasse pas 10 kg.

1. Proposer une donnée de 30 poids qui demande « le moins de calcul ».
2. Proposer une donnée de 30 poids qui demande « le plus de calcul ».
3. La fonction listeCombinaison ci-dessous génère le nombre de combinaison pour nbsac nombre de sac et nbObjet nombre d'objets :

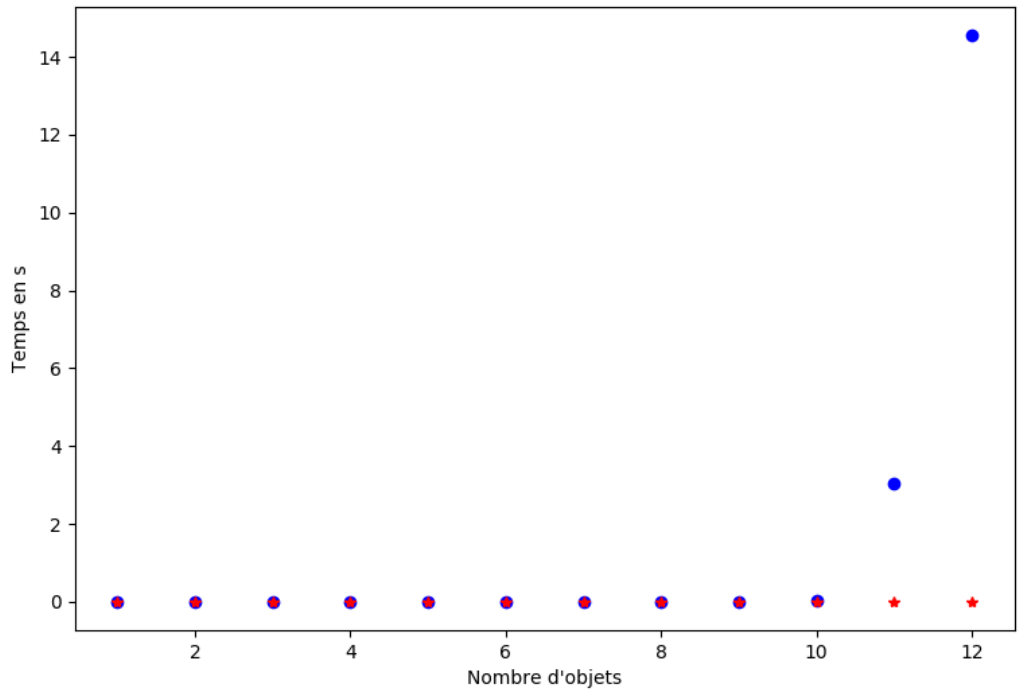
```
def produit2Listes(l, m):
    prod=[]
    for i in l:
        for j in m:
            prod.append(i+j)
    return prod

def puissanceListe(l, n):
    """ Donne les combinaisons avec repetition de n elements de la liste l"""
    if n == 1:
        return l
    else:
        listeAvant=puissanceListe(l,n-1)
        return(produit2Listes(l,tuple(listeAvant)))

def listeCombinaison(nbsac, nbObjet):
    l=[[i] for i in range(1,nbsac+1)]
    return puissanceListe(l,nbObjet)
```

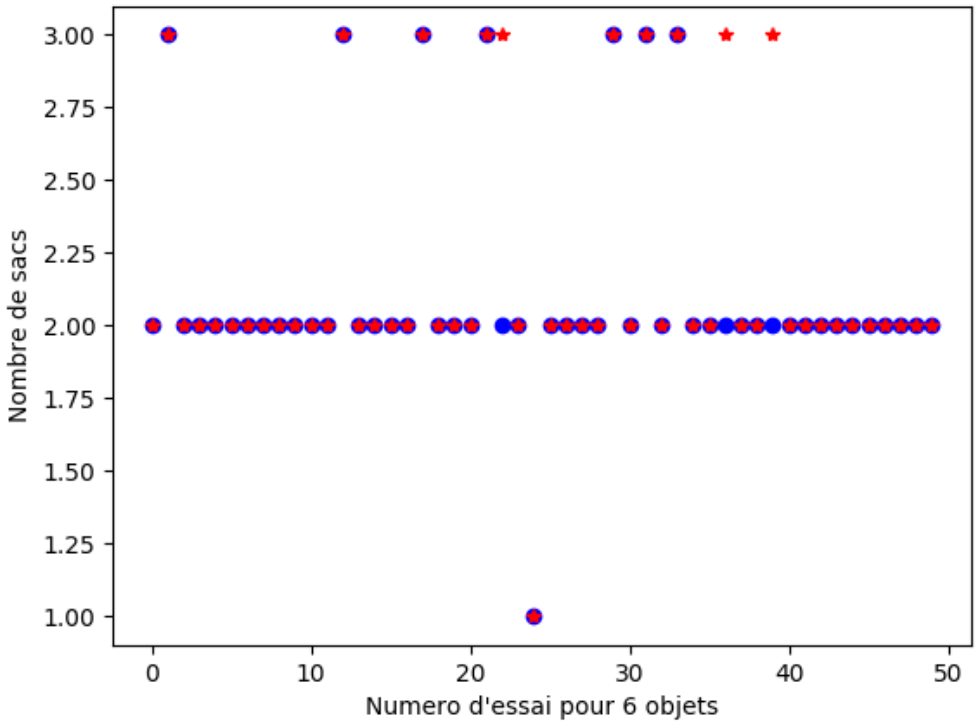
Donner le nombre de combinaison dans les deux cas précédents des questions précédentes. Puis testez-la pour differentes valeur (plus petites).

4. Voici un graphique qui compare l'algorithme FFD (rouge) et un algorithme exact (bleu).



Qu'en pensez-vous ?

5. Bien entendu la question qui se pose alors et de savoir si l'algorithme exact est plus « efficace » au sens que FFD donne des solutions souvent très éloignées de l'optimal. Voici un graphique.



Qu'en pensez-vous ?

6. La fonction rechercheOptimal donne la première solution optimale. Comparer FFD et cet algorithme avec $L = [7,3,8,9,3,9]$.

Pouvez-vous trouver un autre cas où les résultat diffèrent ?

Nous pouvons donc conclure que cette méthode est intérésante car elle fournit toujours des solutions optimales mais qu'elle est peu efficace en terme de temps !

3 D'autres algorithmes

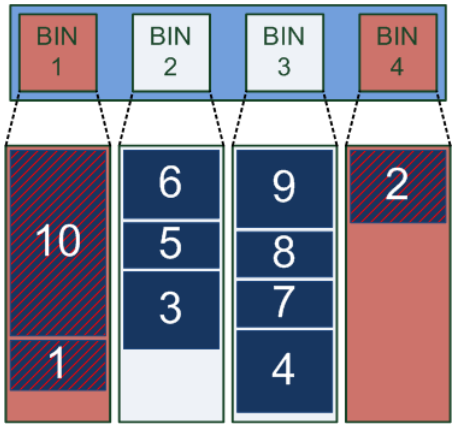
Sachez qu'il existe bien d'autres algorithmes gloutons resolvant ce problème :

— Le Worst Fit Decreasing qui dipose lui aussi les objets les uns après les autres mais avec le critère glouton de mettre l'objet dans le sac le moins lourd à ce moment.

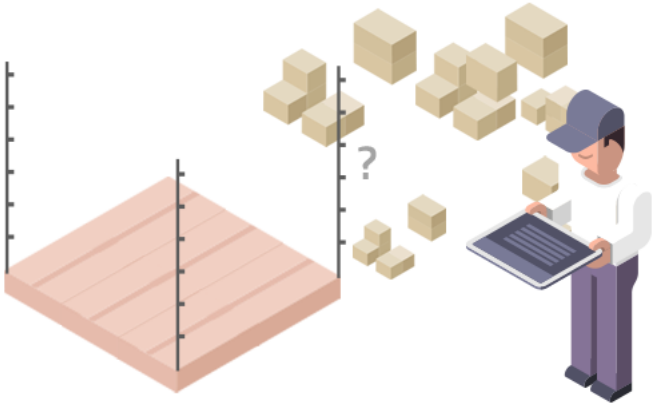
— Le Best Fit Decreasing qui dispose lui aussi les objets les uns après les autres mais avec le critère glouton de mettre l'objet dans le sac le plus lourd à ce moment.

Ce problème de Bin Packing trouve son utilité dans de nombreux domaines :

— en informatique : le rangement des fichiers dans un support informatique, le transport de données.



— dans les transports de marchandises : remplissage de camions avec comme des contraintes sur le poids des marchandises, pas sur le nombre de camions.



Dans tous ces domaines, peu importe que la solution fournie soit optimale (à condition qu'elle n'en soit pas trop loin) à condition qu'elle soit valide et obtenue rapidement !

1 NSI Exercices : Algorithme glouton

Elève

Exercice 1. (Rendu de monnaie)

On dispose d'une quantité suffisante de pièces de 1,2,et 5 dirham. On souhaite rendre N dirham en donnant le nombre minimal de pièces.

1. Définir :
   — l'étape d'itération
   — les choix possibles
   — le critère du « meilleur » choix
   — le test d'arrêt
2. Ecrire sur papier un algorithme glouton reprenant les critères précédents pour répondre au problème.
3. Tester l'algorithme avec $N = 23$ toujours sur une feuille de papier
4. Tester sur papier l'algorithme avec $N = 23$ mais un jeu de pièce de 5 et 6 dirhams.
5. Ecrire cet algorithme en python.

Exercice 2. (Ramasseur de balle)

Un ramasseur de balle doit ramasser un ensemble de balles sur un terrain de tennis et souhaite parcourir la plus petite distance possible.
On propose comme critère de meilleur choix : le ramasseur de balles va jusqu'à la balle la plus proche de lui et la ramasse.

1. Ce critère est il le meilleur choix ? Proposer un schéma pour lequel ce n'est pas le cas.
2. Proposer un critère permettant d'améliorer l'algorithme.

Exercice 3.

On considère le code ci-dessous :

1. définir :
   — l'étape d'itération
   — les choix possibles
   — le critère du « meilleur » choix
   — le test d'arrêt
   Que fait ce code sur le jeu de données :[5,7,8,2,4]

Algorithme Exercice 3.

```
from random import randint
masse_max=10
masses=[5,7,8,2,4]
sac=[[]]

for i in range(len(masses)-1):
    indice_sac=0
    indice=random.randint(0,len(masses)-1)
    while sum(sac[indice_sac])+masses[indice]>masse_max:
        if indice_sac>=len(sac)-1:
            sac.append([])
    indice_sac+=1
    sac[indice_sac].append(masses[indice])
    masses.pop(indice)
print(sac)
print('nb de sac',len(sac))
```

Bilan « bin packing »

Enoncé du problème traité :

On considère un ensemble de sacs identiques et un ensemble d'objets dont on connaît le poids. Sachant que les sacs ne peuvent supporter qu'un poids maximum, combien faudra-t-il au minimum de sacs pour y ranger l'ensemble des objets considérés ?

Traitement du problème :

Pour résoudre ce problème, on a utilisé un **algorithme glouton** qui n'est pas toujours optimal mais qui permet d'obtenir de bons résultats en pratique.

Notre algorithme : **FFT (Fiirst Fit Decreasing)**

1. on trie la liste des n objets par ordre décroissant de poids ;
2. *first-fit*, on prend les objets les uns après les autres et pour chacun, on le place dans le premier sac pouvant le contenir

Remarque : la solution exacte de ce problème nécessiterait un algorithme de complexité d'ordre n^p (ce problème est dit « NP complet », alors que l'algorithme glouton a une complexité d'ordre maximal n^2.

Qu'est-ce qu'un algorithme glouton ?

Algorithme **simple** et **efficace** caractérisé par :

- son mode **itératif** ← Les objets passent de la liste initiale à la liste finale un par un.

- une liste de choix possibles et un ou **des critères de choix** appliqués de manière systématique à chaque itération ← Critère « FFD » : on place chaque objet du plus lourd au plus léger dans le premier sac pouvant le contenir.

- un **test d'arrêt** ← Test sur la taille de la liste initiale contenant les objets qui est amputée à chaque itération de son premier élément

Que renvoie un algorithme glouton ?

En fonction du problème, des données initiales et des critères choisis, un algorithme glouton peut renvoyer :

- une **solution valide ou non**,

- une **solution optimale ou non**

Pour certains problèmes, il existe un algorithme glouton dont on peut faire la preuve qu'il est optimal, pour d'autres problèmes, aucun algorithme glouton n'est optimal.

Des applications du « packing » dans la vie :

En une dimension pour le rangement de fichiers sur un support informatique, pour la découpe de câbles, pour le remplissage de camions ou de containers avec une seule contrainte.