

TP1 Problème de Bin packing

Réaliser par :
Abdelkader TABAA

1 Contents

2	Introduction	2
3	Définition du problème du Bin packing.....	3
4	Méthodes de résolution du problème de Bin packing	5
4.1	Méthodes exactes :	5
4.2	Méthodes approchées :	5
4.2.1	Stratégie Next Fit et Next Fit Decreasing :	5
4.2.2	Stratégie First Fit et First fit decreasing :	7
4.2.3	La stratégie Best fit :	8
5	Les formulations linéaires.....	10
6	Les instances sélectionnées.....	14
6.1	Pour CPLEX :	14
6.2	Pour les heuristiques :	14
7	Code des programmes	16
8	Les graphiques et tableaux de comparaison	21
8.1	Graphiquement en utilisant matplotlib :	21
8.2	Tableaux de comparaison :	23
9	Conclusion	27
10	Bibliographie.....	28

2 Introduction

Le TP consiste à tester plusieurs bin packing algorithmes (de glouton d'approximation) expérimentalement sur les trois Dataset afin de déterminer la qualité des solutions qu'ils produisent en comparant avec les solutions optimaux (m^*), les algorithmes utilisés sont :

1. Next fit (NF)
2. First fit (FF)
3. First fit decreasing (FF)
4. Best fit (BF)
5. Next fit decreasing (NFD)

NB : les algorithmes ci-dessus, sauf Next fit, ont deux implémentations. La version Naïve à une complexité temporelle $O(N^2)$. La version d'implémentation plus rapide utilise un arbre de recherche binaire équilibré (par exemple AVL Tree) et a une complexité temporelle $O(N \log N)$.

3 Définition du problème du Bin packing

L'objectif du problème du Bin packing est de Mettre un ensemble d'objets fixés dans des boites de même volume en utilisant le moins possible de boites.

Tenant l'exemple ci-dessous :

Données :

- N objets à ranger, avec une taille donnée a_i , avec $i \in \{1, N\}$.
- M boites Disponibles, avec la capacité C.

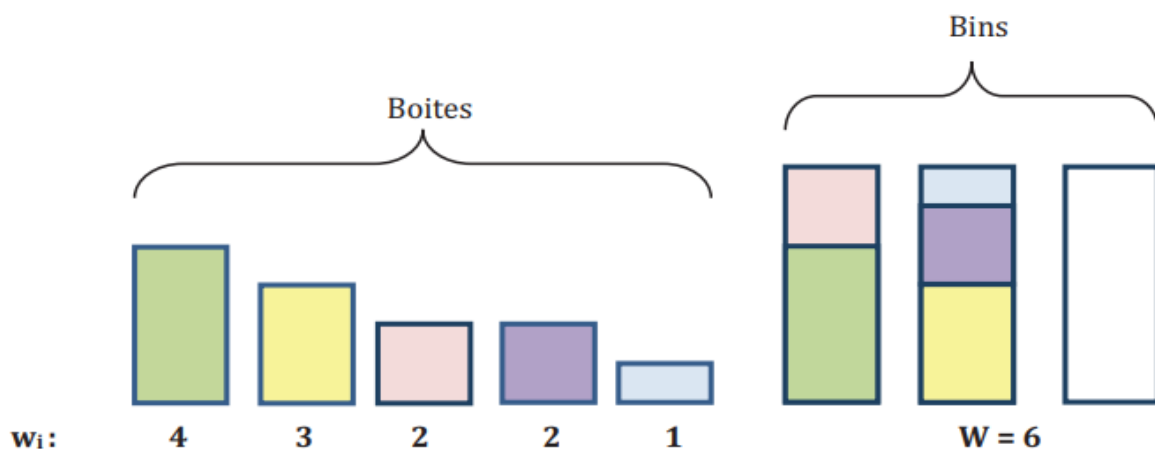
Objectif :

- Affecter chaque objet à une boite.
- Minimiser le nombre de boites utilisées.

Contraintes :

- Respecter la capacité des boites / la taille des objets affectés.

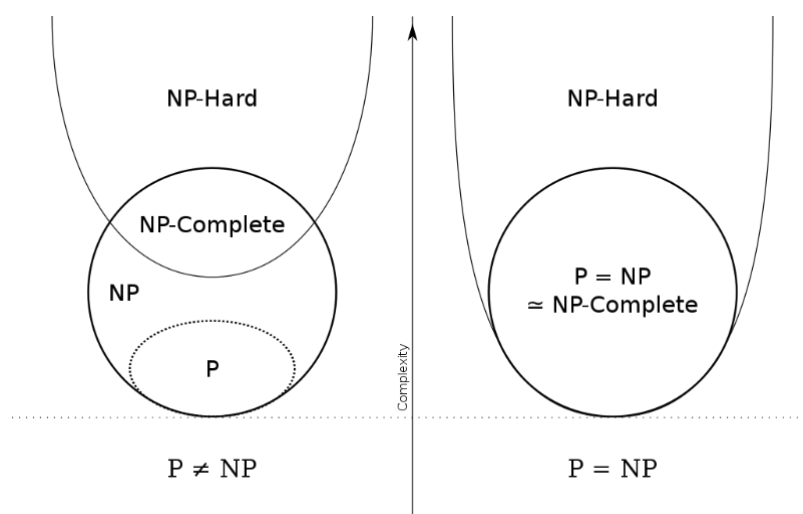
Le problème est NP-Hard (NP-Complete pour la version de décision), il n'y a pas



d'algorithme de temps polynomial connu pour sa solution, et on suppose qu'il n'en existe aucun.

Les applications incluent :

- Chargement de conteneurs comme des camions.
- Placement des données sur plusieurs disques.
- Emballage de publicités dans des pauses radio / TV de durée fixe.
- Stockage d'une grande collection de musique sur des CD.



4 Méthodes de résolution du problème de Bin packing

4.1 Méthodes exactes :

Plusieurs méthodes exactes ont été proposées dans la littérature pour le problème de bin packing et ses variantes.

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles telles que la méthode du simplexe souvent utilisée pour résoudre des programmes linéaires en nombres réels, on trouve aussi la procédure de séparation et évaluation (PSE) ou en anglais Branch and Bound qui est très efficace dans le cas d'un programme linéaire nombres entiers.

4.2 Méthodes approchées :

Dans les problèmes difficiles, comme le cas du problème de bin packing, il est souvent intéressant d'appliquer des heuristiques qui donnent des solutions de bonne qualité dans un temps raisonnable. Les Heuristiques proposées pour le bin packing problème (unidimensionnel) sont :

4.2.1 Stratégie Next Fit et Next Fit Decreasing :

La stratégie Next-Fit (NF) peut être considérée comme le plus simple des algorithmes de résolution du problème de bin packing.

Une fois le premier objet placé, l'algorithme met chaque objet restant dans la boîte du dernier objet placé, S'il n'y a pas de place, il laisse cette boîte et place l'objet courant dans une boîte vide.

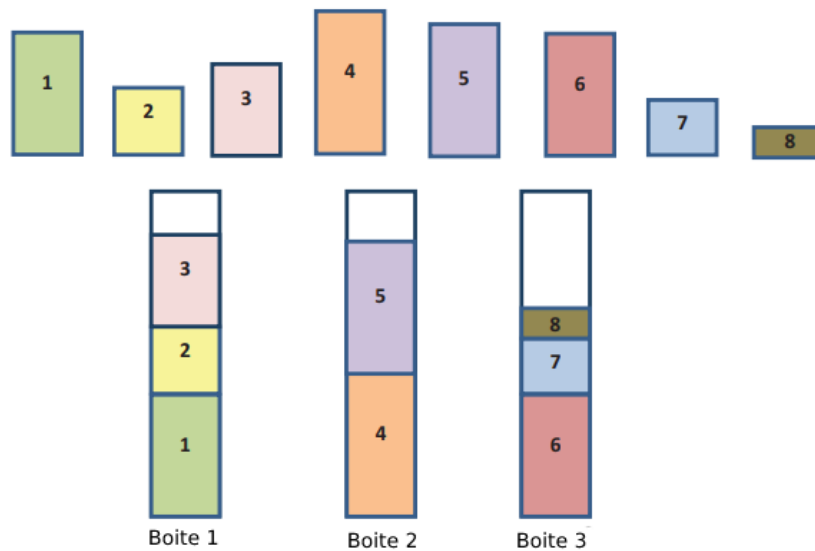


Figure 1: La stratégie NF est appliquée à une série d'objets numérotée de 1 à 8

Le Next Fit Decreasing (NFD) consiste à trier les objets par ordre décroissant de hauteurs et appliquer la stratégie NF pour les ranger.

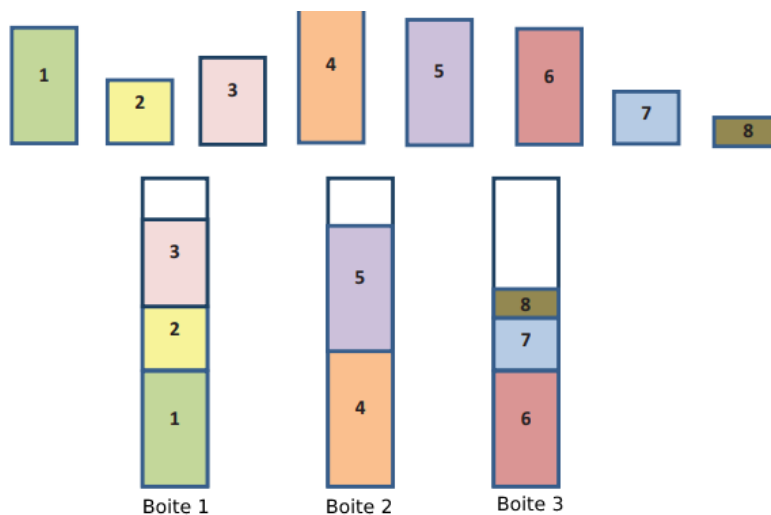


Figure 2: La stratégie NF est appliquée à une série d'objets numérotée

4.2.2 Stratégie First Fit et First fit decreasing :

First fit est un algorithme de résolution du bin packing qui s'applique de la manière suivante :

Pour commencer, on considère une seule boîte, et les objets sont traités selon un ordre donné. Quand il n'y a plus de place dans la première boîte pour ranger l'objet en cours, une deuxième boîte est alors ouverte mais sans fermer la première.

Dans une étape intermédiaire où on dispose de n boîtes ouvertes numérotées de 1 à n selon l'ordre de leur première utilisation, un objet a_i en cours est rangé dans la boîte du plus faible numéro qui peut le contenir.

Dans le cas où aucune boîte ne peut contenir a_i une nouvelle boîte ($n + 1$) est alors utilisée sans fermer les autres. L'ordre selon lequel on traite les objets est crucial pour la qualité de la solution.

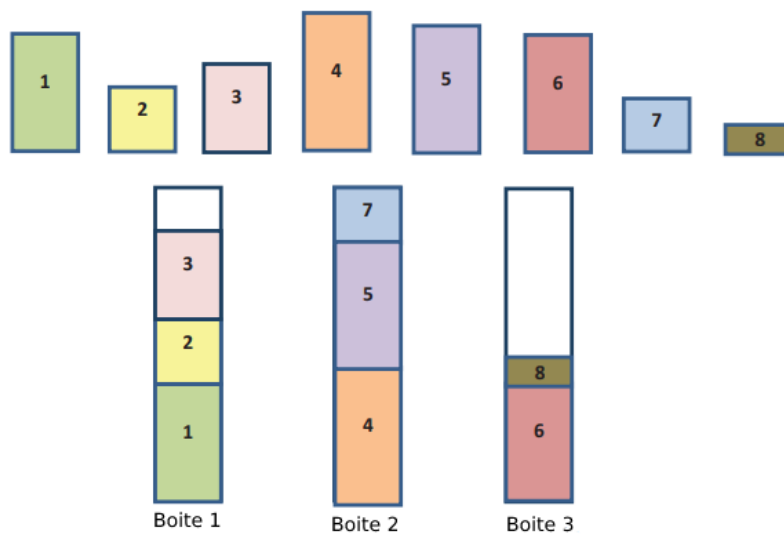


Figure 3: La stratégie First fit appliquée à une série d'objets numérotée de 1 à 8.

First fit Decreasing (FFD) est un choix heuristique qui consiste à trier les objets par ordre décroissant de hauteurs.

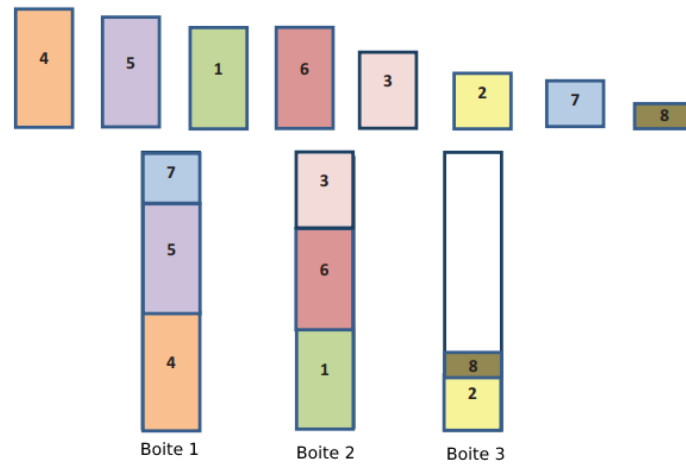


Figure 4: La Stratégie FF est appliquée à une série d'objets numérotée de 1 à 8 préalablement triée par ordre décroissant de hauteur.

4.2.3 La stratégie Best fit :

Les algorithmes Best fit (BF) sont semblables aux algorithmes First fit (FF) dans le sens où ils laissent les boîtes toujours ouvertes.

Cependant dans la stratégie Best Fit, le choix de la boîte dans lequel l'objet ai en cours va être placé dépend des valeurs des gaps (hauteurs non utilisées) présentes dans les boîtes.

Ainsi, un objet ai est alors placé dans la boîte qui présente le moindre gap parmi les boîtes qui peuvent le contenir.

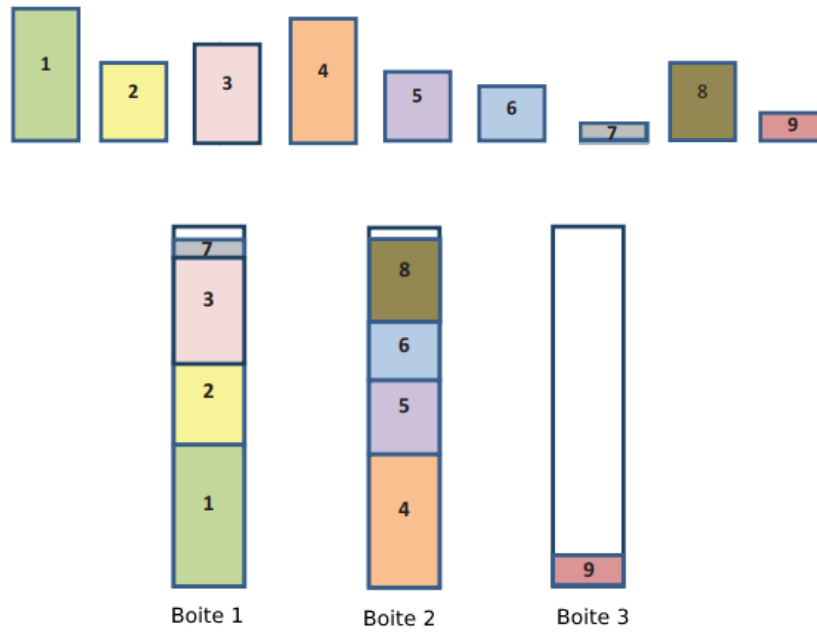


Figure 5: La stratégie Best fit est appliquée à une série d'objets numérotée de 1 à 9.

5 Les formulations linéaires

Le PL de Bin packing :

$$\min \sum_{k=1}^n y_k \quad (1)$$

$$\text{t.q.} \quad \sum_{k=1}^n x_{ik} = 1 \quad (\forall i \in I) \quad (2)$$

$$\sum_{i \in I} w_i x_{ik} \leq W y_k \quad (k = 1, \dots, n) \quad (3)$$

$$y_k \in \{0, 1\} \quad (k = 1, \dots, n) \quad (4)$$

$$x_{ik} \in \{0, 1\} \quad (\forall i \in I, k = 1, \dots, n) \quad (5)$$

Avec y_k : variable entière qui prend la valeur 1 si la boîte k est utilisée et 0 sinon.

x_{ik} : variable entière qui prend la valeur 1 si l'objet i est placé dans la boîte k et 0 sinon.

Les contraintes (2) contrôlent le fait que chaque objet est placé dans une boîte.

Les contraintes (3) vérifient que la somme des tailles des objets placés dans une boîte ne dépasse pas la taille de la boîte.

Sans oublier la contrainte $x_{ik} \leq y_k \quad \forall i = 1, \dots, n, \forall k = 1, \dots, n$ donc $x_{ik} = 1 \Rightarrow y_k = 1$ ça veut dire que si j'ai placé l'objet i dans la boîte k alors il faut que j'active y_k et il faut que je comptabilise la boîte k comme étant une boîte utilisée.

Résolution du Problème en utilisant CPLEX :

Le fichier TP1_binpacking.mod qui contient la modélisation du problème de bin packing (les contraintes et les variables nécessaires, la fonction objective), Ainsi que l'exception implémentée afin de gérer le cas d'erreur si le statut du CPLEX est différent de 1.

```

10 int n = ...;
11 range objet = 1..n; //liste des objets
12 range boite = 1..n; //liste des bins
13 float v= ...; // en cas des capacité réelle
14 float ai[objet]=...;//volume des objets
15
16 dvar boolean x[boite][objet];
17 dvar boolean y[boite];
18
19 minimize sum(i in boite) y[i];
20
21Ⓜsubject to{
22
23Ⓜ forall(i in boite)
24     constraint_1:
25         sum(j in objet) ai[j]*x[i][j] <= v*y[i];
26
27Ⓜ forall(j in objet)
28     constraint_2:
29         sum(i in boite) x[i][j] ==1;
30
31
32
33 }
34Ⓜexecute{
35
36     if(cplex.getCplexStatus()==1){
37         writeln("objet placé dans boite :",x.solutionValue)
38     }
39     else{
40         writeln("erreur");
41     }
42

```

Le fichier TP1_binpacking.dat ci-dessous, pour charger les données des variables (ai = taille des objets et v = la capacité des boîtes, le nombre des objets) depuis un fichier Excel, en utilisant une boucle pour itérer sur tous les instances (Convertir les fichiers. bpp en des fichiers .xlsx) de chaque dataset afin de comparer ses solutions avec les solutions optimal m*.

```

8
9  n=49;
10
11 SheetConnection fichier("Classeur1.xlsx");
12
13 v from SheetRead(fichier,"Feuill1!E3");
14 ai from SheetRead(fichier,"Feuill1!C3:C51");
15
16 main{
17
18 var src = new IloOplModelSource("tpl_binpacking.mod");
19 var def = new IloOplModelDefinition(src);
20
21
22 var iteration=1;
23
24 while(iteration<=5){
25     var opl = new IloOplModel(def,cplex);
26     var filename="data (" +iteration + ")";
27     var data = new IloOplDataSource(filename+".xlsx");
28     opl.addDataSource(data);
29     var details=opl.dataElements;
30     opl.generate();
31     if(cplex.solve()){
32         writeln(filename+"->" +1);
33     }
34     else{
35         writeln(filename+"->" +0);
36     }
37     iteration++
38 }

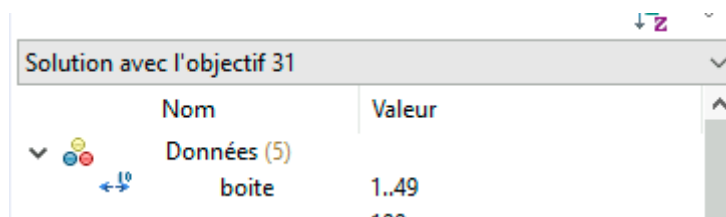
```

Lorsqu'on teste avec les objets de la première instance 'N1C1W1_A' du premier dataset on trouve 25 comme étant une solution optimale.

The screenshot shows the IBM ILOG CPLEX Studio interface. The top pane displays a project tree for 'relaxation_conrig' containing various files like 'p_relaxation.mod', 'tp1_binpacking.dat', 'dataset3.dat', and 'Classeur1.xlsx'. The bottom pane, titled 'Navigateur d...', shows a table of the optimal solution for the objective value 25.

Nom	Valeur
Données (5)	
boite	1..49
v	100

Et pour la deuxième instance on trouve 31, ainsi de suite ...



Nom	Valeur
Données (5)	
boîte	1..49

On remarque que les solutions obtenues en appliquant notre modèle linéaire sur les instances des dataset sont des solutions optimales.

Donc les méthodes avancées offrent des solutions optimales pour un problème.

La relaxation linéaire du problème :

Elle consiste à relâcher les contraintes d'intégralités des variables au lieu d'imposer que les variables dans un ensemble discrète 0 et 1 on les autorisent à prendre des valeurs fractionnaires

$$\begin{aligned}x_{ij} &\geq 0 \\x_{ij} &\leq 1 \\y_j &\geq 0 \\y_j &\leq 1\end{aligned}$$

La relaxation du modèle sous CPLEX peut se faire en utilisant la conversion des variables en FLOAT

```
IloConversion relax_x = IloConversion( env, y, ILOFLOAT );  
model.add( relax_y );
```

Et puisqu'on a élargi le domaine de définition des variables pour prendre aussi des valeurs fractionnaires comme ça on peut avoir l'ensemble des affectations possibles des objets aux boites (puisque on a supprimé les contraintes d'intégrités).

6 Les instances sélectionnées

6.1 Pour CPLEX :

Pour résoudre le problème linéaire sur CPLEX j'ai utilisé les instances en dessous ,j'ai mis les instances sur des fichier Excel pour faciliter le chargement des données sur CPLEX STUDIO en utilisant la fonction SheetConnection.

Dataset1 :

N1C1W1_A N1C1W1_B N4C2W4_A N4C2W4_B N4C3W4_S N4C3W4_T

Dataset2 :

N1W1B1R0 N1W1B2R8 N1W3B1R6 N2W1B1R0 N4W2B3R3 N4W4B3R9

Dataset3 :

HARRD1 HARRD2 HARRD4 HARRD6 HARRD7

6.2 Pour les heuristiques :

Par contre pour le test des heuristiques j'ai utilisé toutes les instances de tous les dataset en se basant sur deux fonctions, la première parcourt le dossier du dataset (bin1data par exemple ou bien bin2data ...) et elle extrait les noms des instances et elle les mis sur une List.

```
def explorer_arbores(dataset):  
  
    path = r'C:\Users\abdel\PycharmProjects\pythonProject1'  
    slash="\"  
    chemin=path+slash+dataset  
    nom = []  
    files = os.listdir(chemin)  
    for name in files:  
        nom.append(name)  
    return nom
```

Ensuite elle vient le rôle de la deuxième fonction qui prend la liste des instances retourner après le premier parcours du dossier du dataset et elle commence à lire chaque fichier et de mettre son contenu dans une liste et de convertir cette liste de liste à une liste normale et de faire un casting ou conversion des éléments string en Int, et comme ça finalement j'ai ma liste qui contient le contenu de l'instance x et je peux l'appliquer sur n'importe algorithme.

```
def lecture_fichier(fichier):  
    with open(fichier, 'r') as filehandle:  
        # pour mon fichier  
        list_of_lists = []  
        for line in filehandle:  
            stripped_line = line.strip()  
            line_list = stripped_line.split()  
            list_of_lists.append(line_list)  
        filehandle.close()  
        merged = list(itertools.chain(*list_of_lists))  
        for i in range(0, len(merged)):  
            merged[i] = int(merged[i])  
        return merged
```


7 Code des programmes

```
import os
import matplotlib.pyplot as plt
import numpy as np
import itertools
```

L'utilisation de la bibliothèque os (operating système) pour profiter des fonctions de parcours des fichiers par exemple ou bien les fonctions qui aide ouvrir et lire un fichier.

Matplotlib pour tracer et afficher des courbes et des histogrammes, des figures.

Numpy pour travailler avec des tableaux.

Itertools utilisé pour itérer sur des structures de données.

```
def fichier_csv(list1):
    #list1=lecteur_metoile(dataset)
    with open('nextfitsolution.csv', 'w') as f:
        for item in list1:
            f.write("%s\n" %item)
```

La fonction fichier_csv est implémenté afin de convertir les solutions ou l'output d'un algorithme en fichier csv, pour effectuer une comparaison avec les tableaux après en utilisant excel.

```

def dessin(liste, liste2, liste3, liste4, liste5):
    ls = []
    ls = lecteur_metoile(dataset)
    #premier algo best fit
    plt.subplot(321)
    plt.plot(liste, "r-", label="solution heuristique")
    plt.plot(ls, "k-", label="optimal")
    plt.grid(True)
    plt.title(' (Best fit) ')
    #deuxième algo next fit
    plt.subplot(322)
    plt.plot(liste2, "y-", label="solution heuristique")
    plt.plot(ls, "k-", label="optimal")
    plt.grid(True)
    plt.title('(Next fit)')
    # troisième algo next fit decreasing

```

La fonction dessin qui prend en entrée 5 liste correspondant aux solutions trouvées par les algorithmes d'heuristiques et qui les affiche sur des figures avec l'affichage des solutions optimales.

```

def lecteur_metoile(dataset):
    if dataset == "bin1data":
        fichier="metoile1.txt"
    if dataset=="bin2data":
        fichier="metoile2.txt"
    if dataset=="bin3data":
        fichier="metoile3.txt"

    nvx=[]
    planetes = open(fichier, 'r')
    lignes = planetes.readlines()
    planetes.close()
    tab = []
    for chn in lignes:
        tab.append(chn.split(','))
    merged = list(itertools.chain(*tab))
    merged = [x.replace('\t', ',').replace('\n', ',') for x in merged]
    merged2 = [s.split(',') for s in merged]
    merged2 = list(itertools.chain(*merged2))
    nvx = [x for x in merged2 if x]
    for i in range(0, len(nvx)):
        nvx[i] = int(nvx[i])
    return nvx

```

Fonction lecture m_etoile qui ouvre le fichier m* qui correspond au dataset entré, car j'en ai besoin pour l'apparaître dans la figure de chaque algorithme.

```
def sol(dataset):  
    ls = explorer_arbores(dataset)  
    v=lecteur_metoile(dataset)  
    my = []  
    final1=[]  
    final2=[]  
    final3=[]  
    final4=[]  
    final5=[]  
    path = r'C:\Users\abdel\PycharmProjects\pythonProject1'  
    slash = "\\ "  
    for i in range(len(ls)):  
        name = path + slash + dataset + slash + ls[i]  
        my = lecture_fichier(name)  
        nv=[my[i] for i in range(2,len(my))]  
        final1.append(best_fit(my[1], nv))  
        final2.append(next_fit(my[1], nv))  
        final3.append(next_fit_dec(my[1], nv))  
        final4.append(first_fit(my[1], nv))  
        final5.append(first_fit_dec(my[1], nv))  
  
    solution=desin(final1,final2,final3,final4,final5)  
    fichier_csv(final2)  
  
    return solution
```

La fonction sol ou bien solution qui contient une variable path et à chaque itération de la boucle elle prend un nom du fichier depuis la liste et le concatène avec le path et un slash pour le lire et applique son résultat sur les algorithmes qui prend le premier élément de la liste qui est la capacité des boîtes, et le reste des éléments qui sont les objets.

```

def best_fit(c, w):
    n=len(w)
    if n == 0:
        return 0

    Bins = [c]
    for i in range(len(w)):
        RC = []
        RCind = []
        for j in range(len(Bins)):
            if w[i] <= Bins[j]:
                RC.append(Bins[j] - w[i])
                RCind.append(j)
        if len(RC) > 0:
            Bins[RCind[np.argmin(RC)]] -= w[i]
        else:
            Bins.append(c)
            Bins[len(Bins)-1] -= w[i]

    return len(Bins)

```

L'algorithme heuristique Best fit.

```

def next_fit(c, w):
    n=len(w)
    if n == 0:
        return 0

    curBin = c
    nBins = 1
    for i in range(len(w)):
        if w[i] <= curBin:
            curBin -= w[i]
        else:
            curBin = c-w[i]
            nBins += 1
    return nBins

def next_fit_dec(c, w):
    n=len(w)
    if n == 0:
        return 0

```

Les algorithmes Next fit et Next fit decreasing

```

def first_fit(c, w):
    n=len(w)
    if n == 0:
        return 0
    Bins = [c]
    for i in range(len(w)):
        nFit = False
        for j in range(len(Bins)):
            if w[i] <= Bins[j]:
                Bins[j] = Bins[j] - w[i]
                break
            if j == len(Bins)-1:
                nFit = True
        if nFit is True:
            Bins.append(c)
            Bins[len(Bins)-1] -= w[i]

    return len(Bins)

def first_fit_dec(c, w):
    n=len(w)
    if n == 0:
        return 0
    wSorted = w.copy()
    wSorted.sort(reverse=True)
    return first_fit(c, wSorted)

```

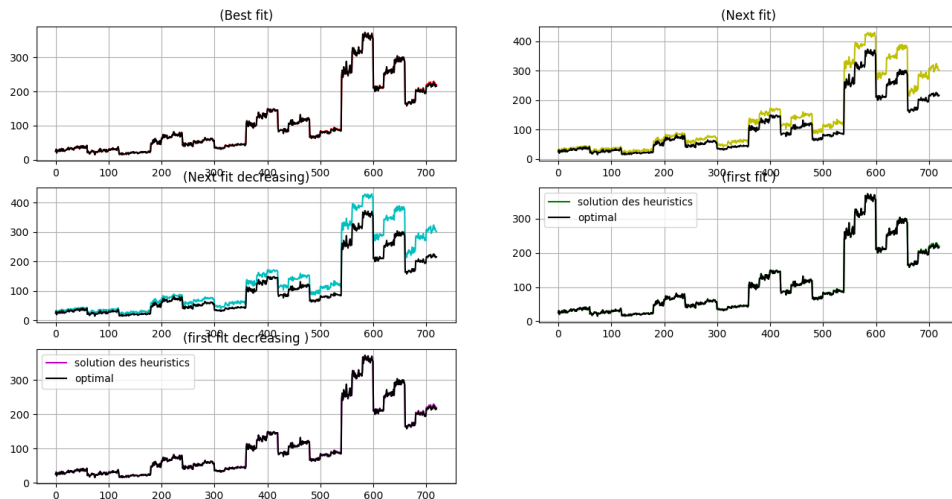
Les algorithmes First fit et First fit decreasing

8 Les graphiques et tableaux de comparaison

8.1 Graphiquement en utilisant matplotlib :

- Pour la première dataset:

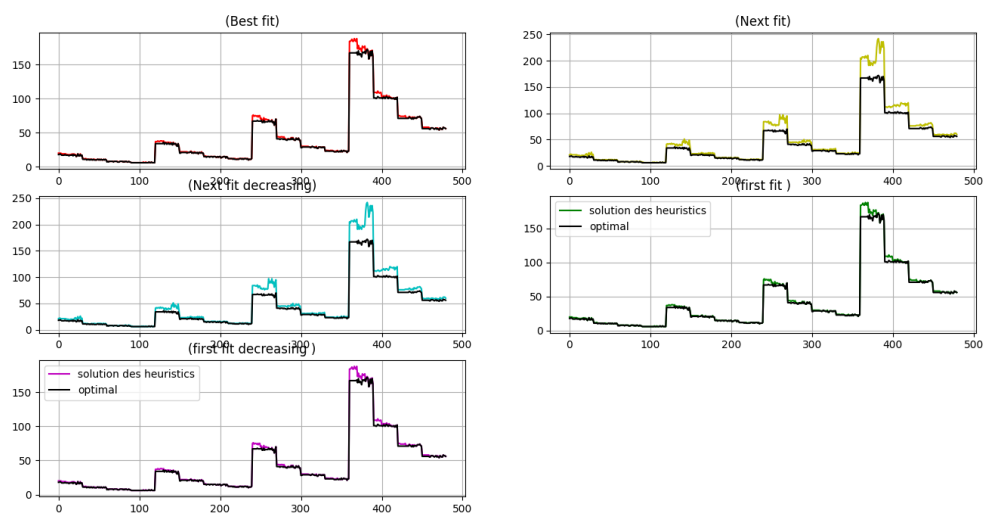
comparaison entre les solution réalisable des algorithmes heuristics et les sol optimal



On remarque que la courbe du Best fit est beaucoup proche de celle des solutions optimaux

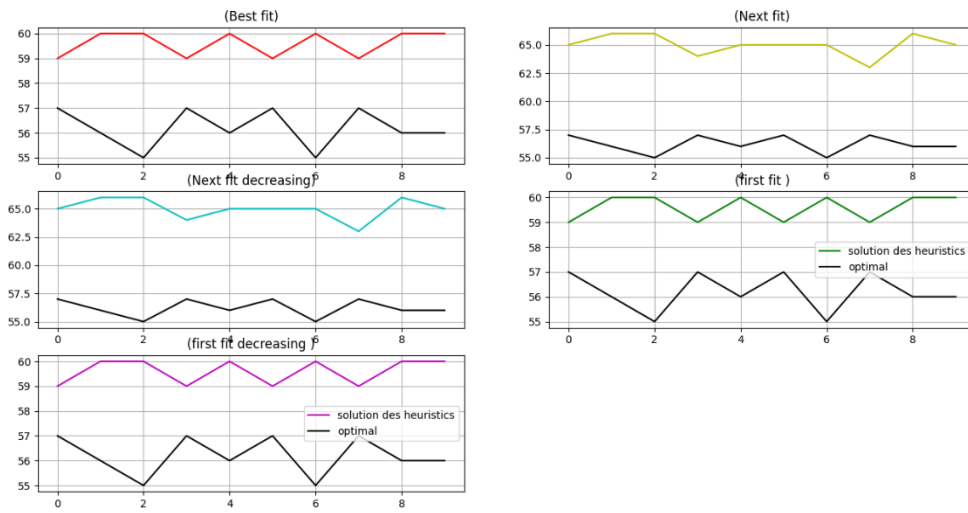
- Pour la deuxième dataset:

comparaison entre les solution réalisable des algorithmes heuristics et les sol optimal



Pour la troisième dataset :

comparaison entre les solution réalisable des algorithmes heuristics et les sol optimal

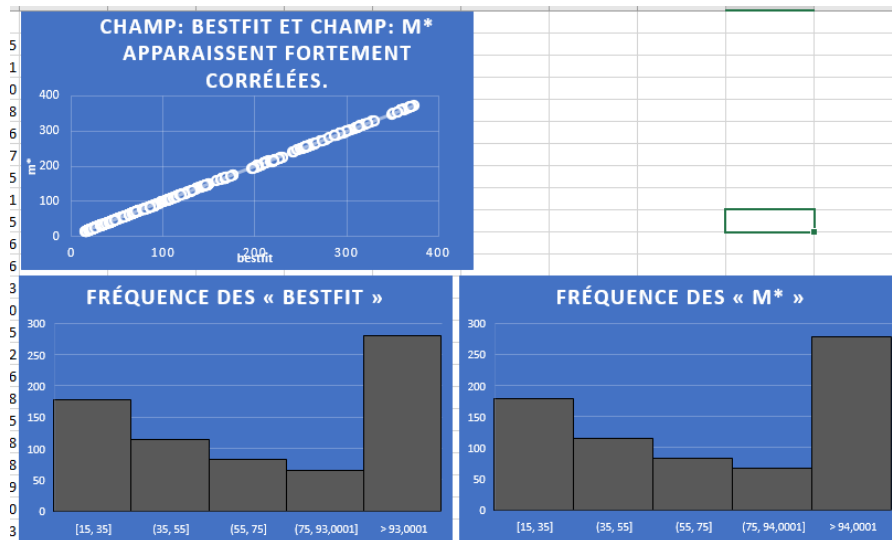


8.2 Tableaux de comparaison :

Après la génération des solutions et les avoir mises dans des fichiers Excel à l'aide de la fonction fichier_csv() que j'ai déjà expliqué ci-dessus, on peut maintenant effectuer des comparaisons entre les solutions réalisables obtenues avec les algorithmes heuristiques et les solutions optimales m^* en utilisant les tableaux Excel.

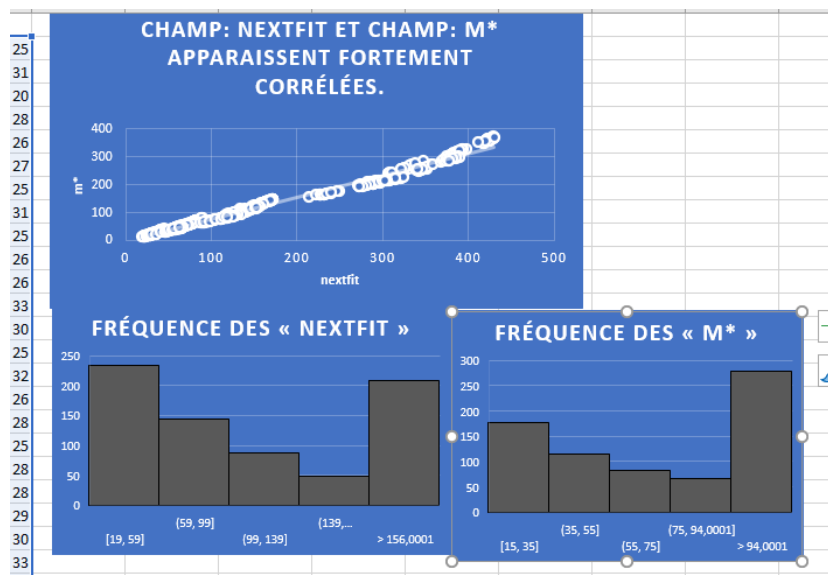
Pour la première dataset :

- Best fit :



Les solutions trouvées par best fit sont un peu proche aux solutions optimales

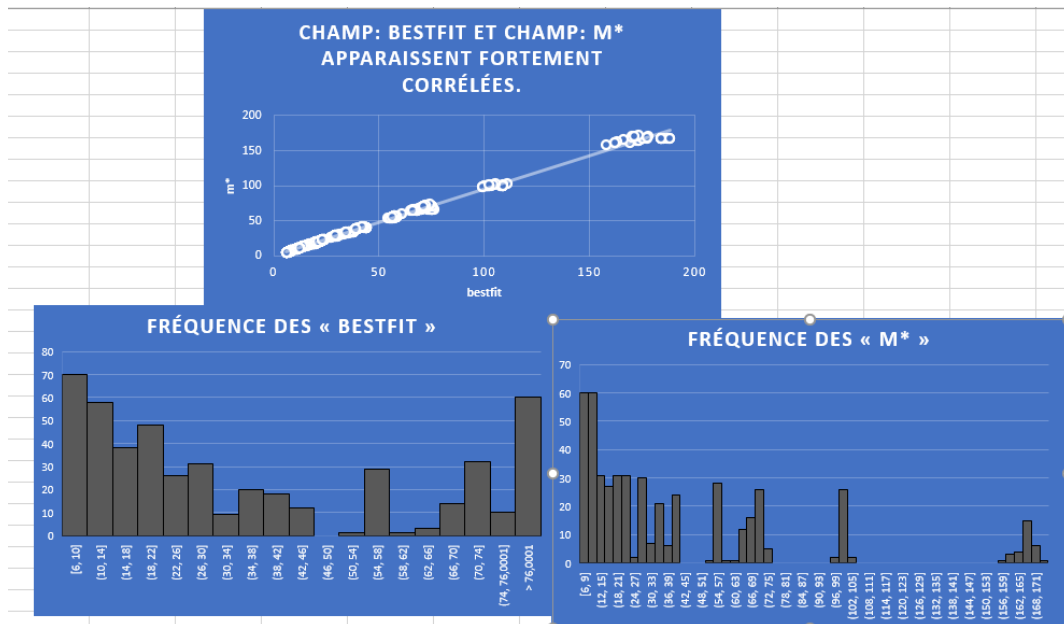
- Next Fit :



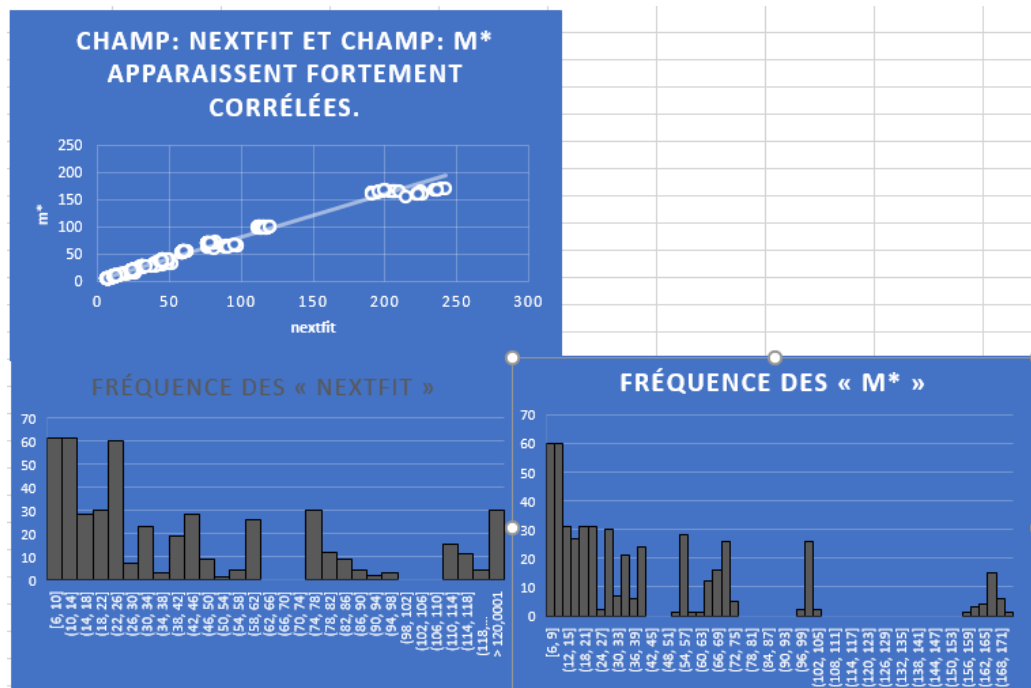
Ainsi de suite pour les autres Algorithmes heuristiques : Next fit decreasing et First fit et First fit decreasing (dans le dossier qui s'appelle comparaison par tableau dossier dataset1).

Pour la deuxième dataset :

- Best fit :



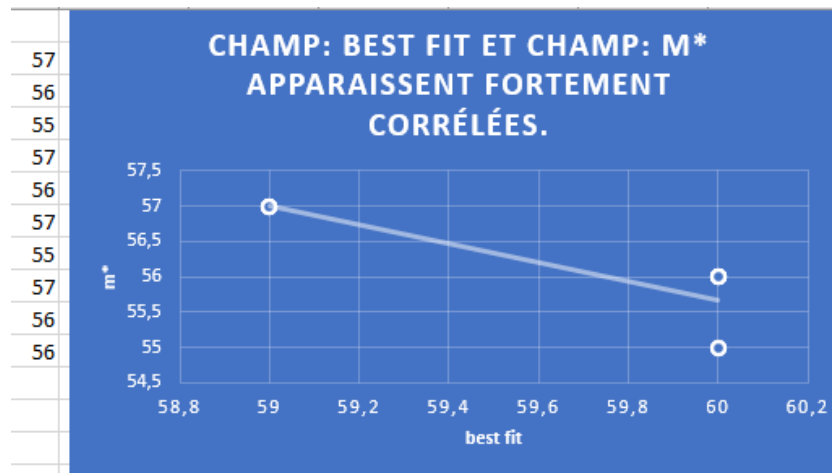
- Next fit :



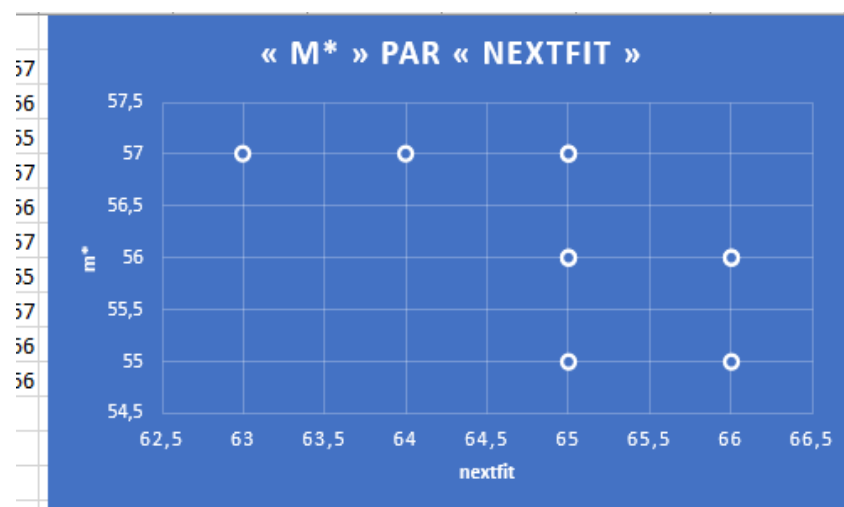
Ainsi de suite pour les autres Algorithmes heuristiques : Next fit decreasing et First fit et First fit decreasing (dans le dossier qui s'appelle comparaison par tableau dossier dataset2).

Pour la Troisième Dataset :

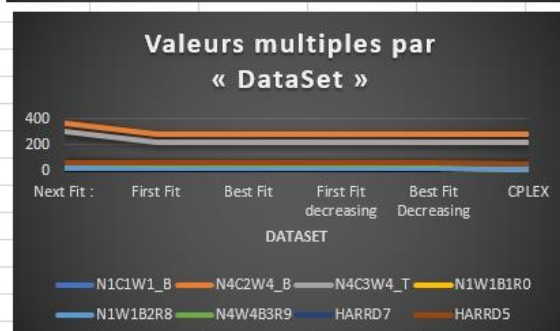
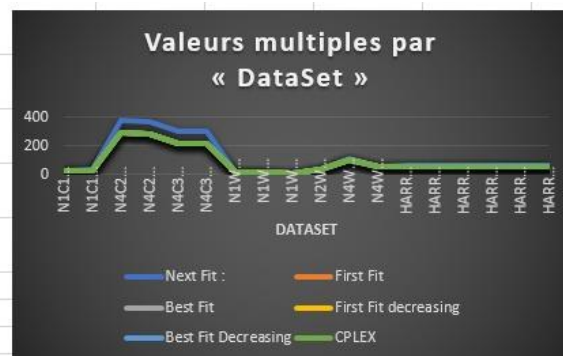
- Best fit :



- Next fit :



DataSet	Next Fit :	First Fit	Best Fit	First Fit decreasing	Best Fit Decreasing	CPLEX
N1C1W1_A	29	25	25	25	25	25
N1C1W1_B	34	31	31	31	31	31
N4C2W4_A	376	293	293	293	293	293
N4C2W4_B	367	281	281	281	281	281
N4C3W4_S	303	220	220	220	220	216
N4C3W4_T	301	220	220	220	220	216
N1W1B1R0	20	20	20	20	20	18
N1W1B2R8	18	17	17	17	17	16
N1W3B1R6	7	8	8	8	8	8
N2W1B1R0	40	37	37	37	37	34
N4W2B3R3	117	101	101	101	101	100
N4W4B3R9	59	56	56	56	56	56
HARRD1	65	60	60	60	60	57
HARRD2	65	60	60	60	60	56
HARRD6	64	60	60	60	60	57
HARRD7	64	60	60	60	60	55
HARRD4	64	60	60	60	60	57
HARRD5	64	59	59	59	59	56



9 Conclusion

Après la comparaison des deux méthodes avec la solution optimale, on remarque que les algorithmes heuristiques offrent des solutions réalisables et approchées avec une convergence rapide, en revanche les méthodes exactes en utilisant simplex offrent des solutions optimales mais qui demandent beaucoup de temps de calcul qui augmente considérablement avec la complexité du sujet à traiter.

En comparant les algorithmes heuristiques entre eux, nous remarquons que l'algorithme best fit offre des solutions meilleur par rapport aux solutions optimales.

10 Bibliographie

https://fr.wikipedia.org/wiki/Probl%C3%A8me_de_bin_packing

https://www.youtube.com/watch?v=qbuMPi44bVQ&t=482s&ab_channel=Dr.HasanJamal

<https://matplotlib.org/stable/index.html>

<https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm>

<http://mistic.heig-vd.ch/taillard/>

<http://fc.isima.fr/~klement/pj/Roadef2015Resume.pdf>