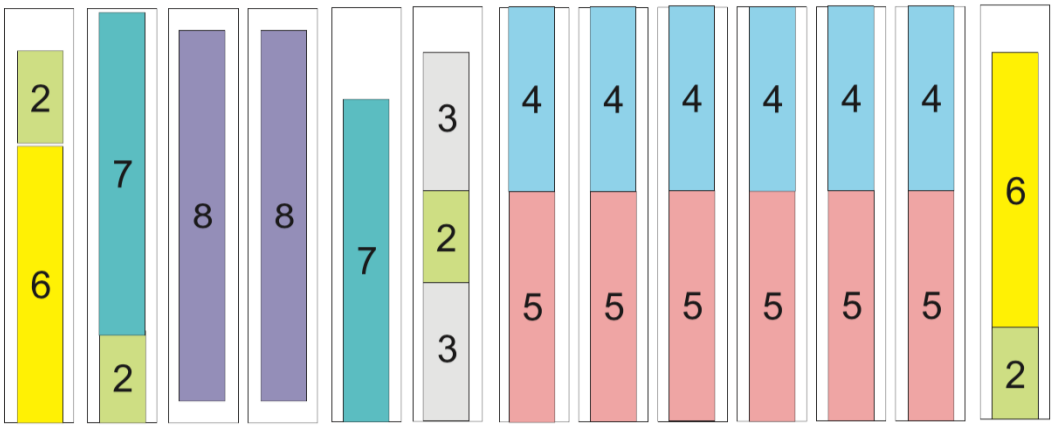
Helloïs BARBOSA 4A - FISA

# Bin Packing 1D

*Optimisation discrète - M. Stéphane BONNEVAY*

*Février - juin 2021*



# Question n°1

Chaque jeu de données se voit attribuer un nombre de maximum de bin correspondant au nombre d'item du problème.

# Question n°2

Chaque jeu de données se voit attribuer un nombre maximum de bin correspondant au résultat de l'algorithme du First Fit Decreasing. Lancez l'exécution de la question n°2 pour avoir les résultats sur tous les jeux de données fournis.

# Question n°3

## a- Trouver la solution optimal du problème "binpack1d_00.txt" [1]

Pour cette question j'ai utilisé la librairie Google OR-Tools. J'ai modélisé le problème linéaire de manière binaire avec deux variables. Une première $x_{ij}$ pour indiquer si l'item i est rangé dans le bin j : cette variable vaudra 1 dans ce cas et 0 sinon.

$$\forall i \in \{1, ..., n\}, \forall j \in \{1, ..., m\}, x_{ij} \in \{0, 1\}$$

Puis une seconde noté $y_j$ valant 1 si le bin est utilisé et 0 sinon. En effet, le problème initial possède un certain nombre de bins qui peuvent se vider au cours de la résolution du problème. Si un bin est mené à être vide, celui-ci sera marqué d'un 0 dans la variable $y_j$.

$$\forall j \in \{1, ..., m\}, y_j \in \{0, 1\}$$

J'ai par la suite défini des contraintes. Notons $p_i$ la taille de l'item $i$ parmis les $n$ items et $S$ la taille d'un bin parmis les $m$ bins. Les premières contraintes permettent de déterminer que la somme des tailles des items ne peuvent pas dépasser la taille totale du bin dans lequel ils se trouvent:

$$\forall j \in \{1, ..., m\}, \sum_{i=1}^{n} p_i \cdot x_{ij} = S \cdot y_j$$

Les secondes contraintes permettent d'affirmer que tous les items sont rangés dans un seul et unique bin.

$$\forall i \in \{1, ..., n\}, \sum_{j=1}^{m} x_{ij} = 1$$

[1] Nommé "sample_00.txt" dans les ressources du projet

Pour finir, j'ai défini la fonction objectif comme suit pour minimiser le nombre de boîtes à utiliser:

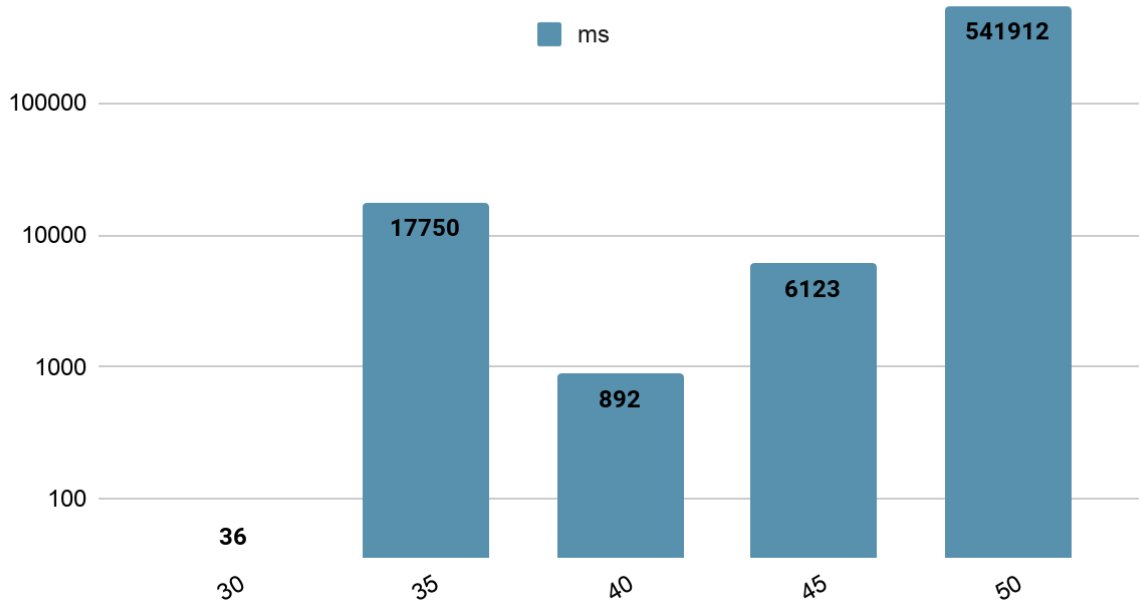
$$min \sum_{j=1}^{m} y_j$$

A l'aide de l'outil OR-Tools, j'ai pu définir les différents paramètres et trouver une solution optimale pour le problème énoncé. La solution est de **13 bins minimum**.

## b- A partir de combien d'item est-il difficile d'obtenir une solution ?

Pour effectuer ce test et avoir des points de comparaisons équivalents et cohérents, il est important de préciser comment est construit le problème initial. Pour cette question c'est l'heuristique First Fit Decreasing qui sera utilisé. Je vais commencer cette phase d'exploration en partant de 30 items et en ajoutant successivement 5 items supplémentaires tirés au hasard dans le fichier initial[2]. La résolution du programme linéaire sera chronométrée.

*Evolution du temps de traitement en fonction du nombre d'item*




[2] Notons que le fichier initial ne comporte que 24 items, pour commencer avec 30, j'ai tiré des tailles d'items au hasard pour le compléter.

En procédant de la sorte on atteint une limite ici à partir de 5O items avec plus de 9 minutes d'exécution. Par ailleurs, ce résultat ne me semble pas très pertinent du fait de la démarche. En effet, selon les combinaisons de départ, une exécution peut durer beaucoup plus de temps qu'à la normale comme on peut le voir ci-dessus avec le test effectué avec 35 items dans le problème (exécution de 17 secondes environ).

J'ai essayé d'exécuter les données fournis et les fichiers "binpack1d_11.txt" et "binpack1d_12.txt" [3] comportant 50 items chacun, n'ont alors respectivement pris que 97 ms et 617 ms. Le fichier "binpack1d_03.txt" [4] comportant lui 60 items, s'exécuta en seulement 47 ms.

Avec toutes ces observations, il est difficile de dire que la limite apparente semble être aux alentours de 50 items. En effet, la méthode empirique nous a montré qu'avec 50 items, certaines combinaisons pouvaient s'exécuter en moins de 100ms tandis que d'autres pouvaient prendre plus de 9 minutes.

## Question n°4 & n°5

Vous trouverez l'Implémentation directement dans le code. Pour donner un exemple, avec le fichier "binpack1d_11.txt" on obtient alors initialement 25 bins pour l'heuristique "First Fit Decreasing" contre entre 26 et 28 pour le "First Fit Random".

En ce qui concerne les différents opérateurs de voisinage, lors du déplacement, si le bin d'origine se retrouve vide, alors ce dernier sera supprimé de la liste des bins de la solution finale. C'est de cette manière que je fais diminuer le nombre de bins dans les solutions. Pour les échanges d'items, on tire simplement des bins au hasard avec des pièces au hasard. On remarque alors qu'on ne peut pas exclusivement utiliser cet opérateur car il ne permet jamais d'améliorer la solution en supprimant des bins.

## Analyse des paramètres des méthodes étudiées

Avant d'analyser plus en détail le paramétrage des valeurs d'entrées des méthodes du recuit simulé et de la recherche tabou, il faut tout d'abord préciser que les opérateurs de voisinage choisis lors de la phase de dégradation de la solution courante sont tiré de manière aléatoire afin de rendre cette génération la plus diverse possible.

[3] Respectivement nommé "sample_07.txt" et "sample_08.txt" dans les ressources du projet
[4] Nommé "sample_03.txt" dans les ressources du projet

A noter aussi que l'heuristique utilisé ici pour générer la solution initiale est le First Fit Random. Voulant parcourir un paysage étendu, j'ai décidé de laisser de côté ici l'heuristique First Fit Decreasing qui donne de bonnes solutions trop rapidement. Avec l'heuristique choisie je vais pouvoir dégrader et améliorer des solutions initiales qui sont plus diverses.

Pour étudier les deux méthodes, j'ai réaliser des batteries de test en suivant la méthode suivante:

- Appliquer 30 fois l'algorithme à tester sur les treizes fichiers sources fournis pour le TP
- Récupérer à chaque itération la liste des résultats trouvés
- Calculer la moyenne, la variance et l'écart-type des résultats obtenus
- Calculer la moyenne des écart-types de la batterie de test

De cette manière, je vais pouvoir comparer les différents paramétrages via la moyenne des écart-types qui représente la dispersion moyenne d'un paramétrage sur tous les fichiers sources de la batterie de test.

## Question n°6

Lors des tests de la méthode du recuit simulé, je vais donc pouvoir faire varier la température initiale, le nombre d'itération par température ainsi que le facteur de décroissement

Commençons d'abord notre étude en faisant varier la température initiale. Comme on peut le voir dans l'annexe n°1 c'est lorsque **la température initiale vaut 100** que j'obtiens le meilleur résultat. Je vais donc paramétrer mon recuit simulé avec cette variable pour les prochaines batteries de test. Pour cette seconde batterie de test j'obtiens dans l'annexe n°2 le meilleur résultat avec **100 itérations par température**. J'ai fini la recherche de paramétrage avec les deux paramètres sélectionnés précédemment. L'annexe n°3 indique que le **facteur de décroissance optimal serait 0.95**. J'aurai pu choisir 0.98 mais la dispersion ne variant pas et la justesse des résultats non plus, j'ai opté pour un paramétrage permettant d'obtenir des résultats plus rapidement.

Pour le recuit simulé j'ai donc gardé le paramétrage suivant:

- température initiale: 100
- température finale: 0.0001
- itération par température: 100
- facteur de décroissance: 0.95

Avec ces paramètres on va avoir une convergence suffisamment rapide et avec une bonne précision.

## Question n°7

Lors des tests de la méthode de la recherche tabou, je vais faire varier le nombre maximum de pas à réaliser dans la recherche, les nombres de voisins à générer pour chaque pas ainsi que la taille de la liste tabou. En effet, le nombre de voisins à générer à chaque pas est ici un paramètre, car, trouver la liste exhaustive de tous les voisins de la solution courante à chaque pas, mènerait à une explosion combinatoire et les problèmes ne seraient pas resolvables en temps raisonnable.

Commençons d'abord notre étude en faisant varier le nombre maximum de pas. Comme on peut le voir dans l'annexe n°4 c'est lorsque le nombre maximum de pas vaut 1000 que j'obtiens le meilleur résultat. En revanche, les résultats obtenus avec **500 pas maximum** sont similaires et prennent moitié moins de temps à être généré. Je vais donc sélectionner un nombre de pas maximum de 500 ici. Je vais donc paramétrer ma recherche tabou avec cette variable par la suite. Pour cette seconde batterie de test j'obtiens dans l'annexe n°5 le meilleur résultat avec **200 voisins générés par pas**. J'ai fini la recherche de paramétrage avec les deux paramètres sélectionnés précédemment. L'annexe n°6 indique que la **taille de la liste tabou optimale serait de 2 ou 3**. Je vais ici sélectionner une taille de 3 car cela permet d'avoir une meilleure précision en explorant plus le paysage en un temps significativement identique qu'avec une liste tabou de taille 2.

Pour la méthode de recherche tabou, j'ai donc gardé le paramétrage suivant:

- nombre de pas maximum: 500
- nombre de voisins générés par pas: 200
- taille de la liste tabou: 3

## Question n°8

D'après l'étude que j'ai réalisée précédemment[5], j'ai obtenu des paramètres optimaux pour l'ensemble des jeux de données fournis pour chaque méthode de résolution de problème d'optimisation. Parmis ces deux méthodes il serait plus intéressant ici d'utiliser la méthode du recuit simulé qui permet de trouver de bons résultats plus rapidement et plus efficacement que la méthode tabou.

La dispersion moyenne des résultats obtenus pendant mon étude montre que la méthode tabou donne des résultats finaux un peu plus dispersés que la méthode du recuit simulé. Si on regarde en parallèle les résultats minimaux trouvés on voit qu'il n'y a pas de différence significative d'une méthode à l'autre. Pour finir, que nous utilisions le recuit simulé ou la recherche tabou, nous obtenons en moyenne de bon résultats.

Pour conclure, il est important de noter que j'ai cherché ici un paramétrage pour chaque méthode de résolution de problème qui devrait s'adapter au maximum à tous les cas de situation présent dans les jeux de données fournis. La raison pour laquelle je choisis le recuit simulé ici et que pour des résultats quasiment identique, le recuit simulé prendra ici en moyenne moitié moins de temps que la méthode de recherche tabou. A noter aussi, que pour effectuer un traitement à l'aide de ces méthodes, il est nécessaire de trouver un paramétrage adapté à chaque problème et qu'il n'en existe pas d'optimaux permettant de répondre à toutes les problématiques. Il faut faire du cas par cas afin de répondre convenablement à chaque problème d'optimisation.

[5] Respectivement l'annexe n°3 et l'annexe n°6