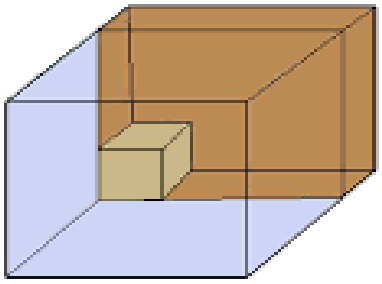
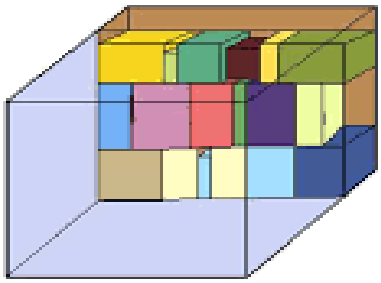
Problème du Bin Packing



Choix de la couche par une boîte



Remplissage de la couche

ADOUANI Mohtadi

DIAKITE Lamine

Master 1 informatique

Année 2021/2022

Dans d’un tp sur le bin packing en optimisation des systèmes qui a pour objectif de maximiser le nombre d’objets qu’on peut ranger dans une (ou plusieurs) boîte(s).
Nous allons essayer d'étudier ensemble les performances de l’algorithme exacte grâce à Cplex et à quelques graphiques d'heuristiques.
Nous allons essayer d'implémenter plusieurs méthodes dont une méthode exacte et une méthode d'heuristiques.

Méthodes exactes

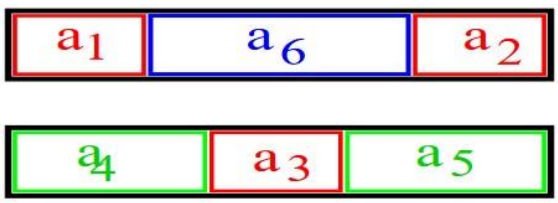
Nous avons implémenté un programme en C qui utilise la bibliothèque Cplex, cet algorithme résout le problème en donnant un résultat exact. Son efficacité est donc la même que pour l’algorithme de résolution vu en cours.

Cette méthode n’est pas très efficace pour des ensembles très grands et pour certaines configurations. C’est-à-dire que la solution optimum pour ce problème ne peut être trouvée qu'après avoir essayé toutes les combinaisons possibles. C'est donc un problème difficile. Cependant, si le nombre de boîtes augmente au fur et à mesure, le nombre d'itérations augmentent aussi au point qu'il ne peut pas être résolu en temps polynomial même par l'ordinateur le plus rapide possible qui puisse exister à nos jours. Mais alors, avec quelques suppositions, ces genres de problèmes peuvent être résolus par des algorithmes heuristiques en un temps presque polynomial et fournir une solution près de l'optimum.

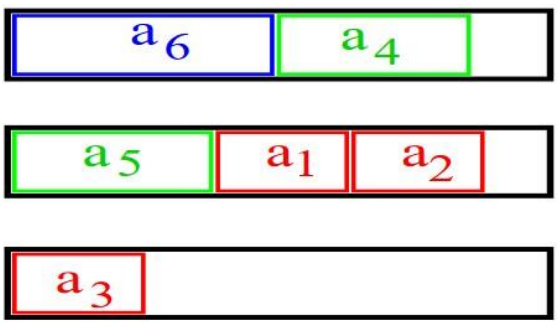
Méthodes heuristiques

La méthode heuristique est plutôt efficace lorsque nous ne cherchons pas une valeur exacte (optimale). Il existe différents degrés d'optimalité pour les deux algorithmes que nous avons implémentés.

Solution obtenue par la méthode exacte



Solution obtenue par First-fit



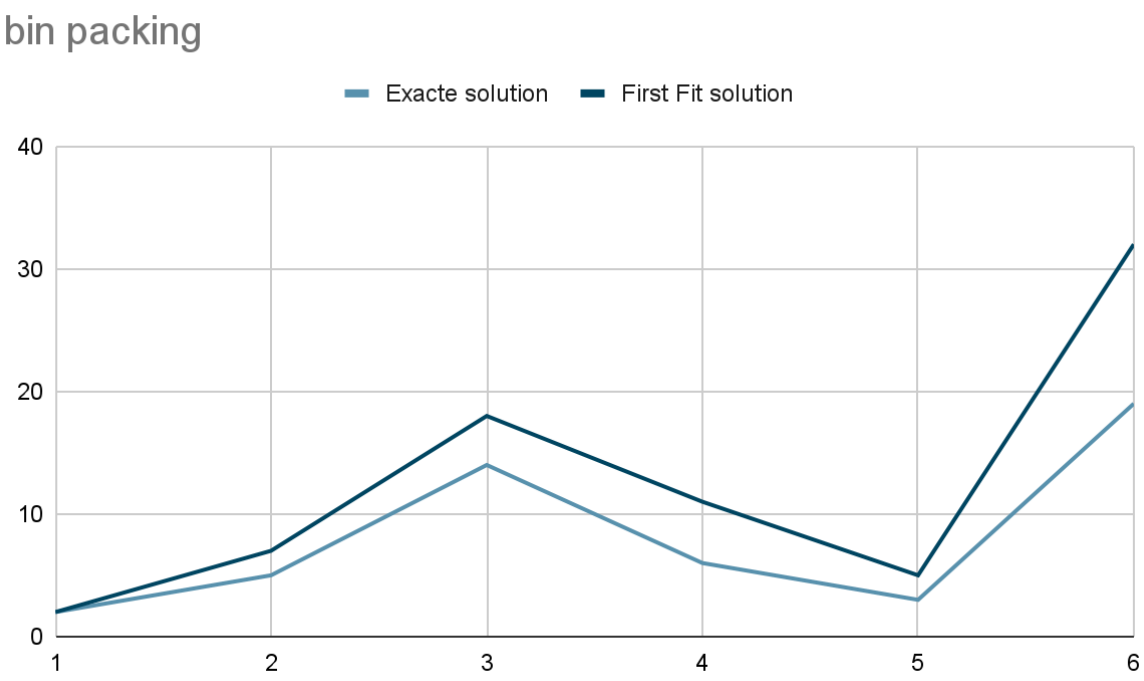
$$c = 8, w(a_1) = w(a_2) = w(a_3) = 2, w(a_4) = w(a_5) = 3.$$

Le problème du Bin Packing(remplissa

Nous avons implémenté deux algorithmes.

First-fit bin packing

Généralités :
First-fit en bin packing : Son entrée est une liste d'éléments de différentes tailles. Sa sortie est un emballage - une partition des articles en casiers de capacité fixe, de sorte que la somme des tailles des articles dans chaque casier soit au plus la capacité. Idéalement, nous aimerions utiliser le moins de bacs possible, mais minimiser le nombre de bacs est un problème NP-difficile.
Dans notre cas :
Cet algorithme peut être vu sous différentes formes. Il peut soit vérifier un à un les objets restants pour savoir lequel doit être mis dans l'espace restant.
Nous pouvons aussi modifier l'entrée en faisant un tri par exemple.
Ces modifications sont une optimisation mineure sur l'algorithme.
Nous pouvons améliorer notre algorithme en essayant plusieurs combinaisons à défaut d'utiliser plus de temps et de ressources d'exécution. Ici ce n'est pas le but car on aurait pu utiliser la méthode
Preuve de Ullman
Optimality de First Fit ≤ 1.7 Solution optimale + 3
Quelques expériences :



Nous pouvons voir l'écart entre les deux algorithmes mais ça reste une solution plutôt bien dans certains cas.

Next-fit bin packing

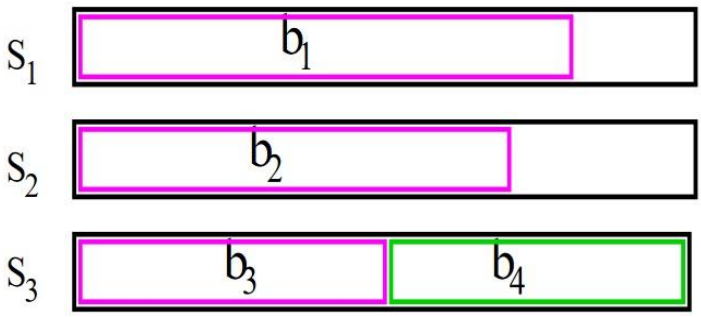
Généralités :
Next-fit est un algorithme en ligne pour le bin packing. Son entrée est une liste d'éléments de différentes tailles. Sa sortie est un emballage - une partition des articles en casiers de capacité fixe, de sorte que la somme des tailles des articles dans chaque casier soit au plus la capacité. Idéalement, nous aimerions utiliser le moins de bacs possible, mais minimiser le nombre de bacs est un problème NP-difficile.
Dans notre cas :
Cet algorithme est beaucoup moins puissant que dans la littérature il ne fait pas ce qu'il devrait faire, mais nous l'avons laissé quand même.
Borne inferieure

$$NF(L) = 2 \cdot \mathrm{OPT}(L) - 2$$

Cet algorithme est plus performant lorsque nous itérons sur les objets.

L'algorithme First Fit Decreasing retourne une solution à un facteur 2 de l'optimal.

Preuve :
CAS Où w(bn) ≥ c/2 : l'algorithme glouton est optimal



Au plus 2 éléments par sac.

EXEMPLE: $c = 10, w(b_1) = 8, w(b_2) = 7, w(b_3) = 5, w(b_4) = 5$

Si on considère n comme le nombre de toutes les boîtes à être placées dans l'espace de stockage et k comme le type de différentes boîtes de tailles différentes, la durée d'exécution pour le cas le pire de cette exécution est exprimé de la manière suivante :

T(n) = n (nk)

T(n) = kn2 où k est une constante.

Au final, la complexité de cet algorithme est O(n2).

Conclusion :

La solution exacte reste le meilleur algorithme pour résoudre un problème exact, cela dit des alternatives d'heuristiques parmi celles citez ici sont une bonne alternative pour éviter une complexité trop élevée. Lorsque nous avons un très grand problème c'est bien d'essayer ces algorithmes d'approximations.

Bibliographie :

http://www.wattebled.fr/pages/memoire.pdf

https://www.lri.fr/~jcohen/documents/enseignement/binpacking.pdf

https://members.loria.fr/LMendoza/docs/memoire2.pdf

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