Nantes Université

Branch-and-bound pour le bin-packing unidimensionnel

Optimisation discrète et combinatoire

Table des matières

Juanfer Mercier – Tobias Sanvoisin

Université de Nantes — UFR Sciences et Techniques
Master informatique parcours "Optimisation en Recherche Opérationnelle"

Année académique 2022-2023

1 Présentation du problème

1.1 Introduction

Le bin packing est un problème d'optimisation NP-Difficile, qui consiste à ranger des objets de taille variable dans un nombre minimum de boîtes ou de conteneurs de taille fixée (appelé plus communément "bins"). Il existe plusieurs variantes du problème de bin packing, notamment sur le nombre de dimensions utilisées pour le problème : bin packing unidimensionnel (BB1D), bi-dimensionnel (BB2D) ou tri-dimensionnel (BB3D). Voici quelques exemples d'application du bin packing en fonction du nombre de dimensions :

- BB1D : l'optimisation du stockage sur les objets numériques (e.g. CD, clés USB)
- BB2D : la découpe de matériaux
- BB3D : le remplissage d'un conteneur ou d'une valise avant de partir en voyage.

Dans notre cas, nous allons surtout nous intéresser au problème unidimensionnel (BB1D) et à une méthode de résolution de celui-ci : le branch-and-bound (BnB ou B&B).

1.2 Le bin packing unidimensionnel

Le problème de bin packing 1D (BB1D) est un problème dans lequel :

- n est Le nombre total d'objets à considérer,
- i est l'indice d'un objet avec $i = 1, \ldots, n$,
- w_i représente l'espace occupé par l'objet i,
- C est la capacité d'un bin

On supposera aussi que $\forall\, i \in 1, \ldots, n\,,\ w_i \in \mathbb{N}$, $C \in \mathbb{N}$ et $w_i \leqslant C$. De plus, les bins sont disponibles en quantité suffisante et présentent tous la même capacité. Aussi, les objets sont indivisibles.

Résoudre le BB1D consiste à ranger chaque objet dans un bin de manière à utiliser le moins de bin possible. On supposera dans la suite que dans toute solution réalisable, les valeurs d'indices de bins les plus petites indiquent les bins utilisées, c'est-à-dire si les m bins sont indicés par $j \in 1, \ldots, m$, et que l'on associe à chaque bin une variable y_j qui prend la valeur 1 si le bin j est utilisé (0 sinon), on aura $y_j \geqslant y_{j+1}$ pour $j \in 1, \ldots, m-1$.

1.3 Instance didactique

Considérons l'instance présentée Table 1 pour le problème BB1D. Étant donné le nombre relativement restreint de bins, il est envisageable de construire une solution admissible de bonne qualité pour l'instance didactique. Par exemple, on peut poser $u_i = \dfrac{w_i}{C}$ des utilités pour chaque objet $i \in 1, \ldots, 7$ (voir Table 2).

n		w_1	w_2	w_3	w_4	w_5	w_6	w_7		C
7		2	3	3	3	4	4	5		6

TABLE 1 – Instance didactique comprenant 7 objets et de m bins de capacité $C = 6$

u_1	u_2	u_3	u_4	u_5	u_6	u_7
$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{5}{6}$

TABLE 2 – Utilités pour chaque objet $i \in 1, \ldots, 7$

Ensuite, nous pouvons placer les objets dans des bins de la manière suivante :

1. Placer l'objet de plus petite utilité dans un bin et essayer de le mettre dans le premier bin disponible
2. Parcourir la liste des objets dans l'ordre décroissant des utilités et placer le premier objet (différent de l'objet sélectionné à l'étape précédente) qui rentre dans le bin sans en dépasser la capacité. S'il n'y a pas d'objet différent de celui choisie à l'étape 1 qui rentre dans le bin sans en dépasser la capacité alors l'objet sélectionné à l'étape 1 est placé tout seul dans un bin.

Avec cette méthode, nous obtenons :

objets $o_i \; \forall i \in 1, \ldots, 7$	o_1 o_6	o_2 o_4	o_3	o_5	o_7
bins $b_j \; \forall j \in 1, \ldots, 5$	b_1	b_2	b_3	b_4	b_5

TABLE 3 – Solution admissible du BB1D pour l'instance didactique

Il faut en tout 5 bins pour ranger ces objets avec cette méthode et 5 est une borne primale (borne supérieure) sur le nombre de bins nécessaires : $m = 5$ et $\forall\, j \in 1, \ldots, 5\,, y_j = 1$.

1.4 Borne inférieure sur le nombre de bins nécessaires

Dans la perspective d'encadrer la valeur optimale du BB1D, disposer d'une borner duale est crucial. En supposant que les objets sont divisibles et qu'ils peuvent être scindés, on peut remplir les bins jusqu'à atteindre leur capacité (voir TABLE 4). On suppose ici que le problème dual du BB1D est obtenue par une relaxation linéaire de la contrainte d'indivisibilité des objets.

objets $o_i \; \forall i \in 1, \ldots, 7$	o_1 o_6	o_2 o_4	o_3 $\frac{3}{4}o_5$	o_7 $\frac{1}{4}o_5$
bins $b_j \; \forall j \in 1, \ldots, 4$	b_1	b_2	b_3	b_4

TABLE 4 – Solution admissible du dual du BB1D pour l'instance didactique

Soit u la borne supérieure obtenue (borne primale) et l la borne inférieure obtenue (borne duale), on obtient un encadrement de la solution optimale au BB1D : $l \leqslant m \leqslant u$ avec $l \in \mathbb{N}$ et $u \in \mathbb{N}$. En effet, en considérant que les objets sont indivisibles il est possible de remplir les bins jusqu'à capacité mais si l'on scinde un objet alors l'admissibilité de la solution pour le BB1D primal est perdue.

Ainsi, il faut au moins l bins pour obtenir une solution admissible pour le BB1D primal (dans le cas trivial où il est possible de faire rentrer tous les objets dans des bins jusqu'à capacité et sans scinder les objets, on aura $m = l$ et la solution duale sera admissible pour le problème primal). Pour obtenir une solution primale admissible à partir d'une solution duale non réalisable, il suffit de sélectionner les fractions d'objets scindés pour reconstruire les objets qui ont été scindés et placer les objets reconstruits dans des bins.

Pour l'instance didactique, il est possible d'établir la valeur de la solution optimale à partir de l'encadrement $4 \leqslant m \leqslant 5$. En effet, la solution duale admissible obtenue n'est pas primale admissible (a priori, $m \geqslant 4$) mais en reconstruisant l'objet w_5 et en le plaçant dans un nouveau bin, on obtient une solution primale admissible optimale avec $m = 5$.

1.5 Modélisations MILP

Afin d'écrire deux modèles représentant le BB1D, nous allons poser les éléments permettant de le modéliser.

Indices

- i : l'objet d'indice i
- j : le bin d'indice j

Données

- n : le nombre d'objets à ranger dans les bins, avec $i \in 1, \ldots, n$
- m : le nombre de bins, avec $j \in 1, \ldots, m$
- w_i : l'espace occupé par l'objet i avec $i \in 1, \ldots, n$
- C : la capacité d'un bin

Variables de décision

- $x_{ij} = \begin{cases} 1 \text{ si l'objet } i \text{ est placé dans le bin } j \\ 0 \text{ sinon} \end{cases}$
- $y_j = \begin{cases} 1 \text{ s'il y a un objet rangé dans le bin } j \\ 0 \text{ sinon} \end{cases}$

Modèle P1

$$
\left[\begin{array}{lllll}
\min \quad z = & \displaystyle\sum_{j=1}^{m} y_j & & & (1.0) \\
s.c. & \displaystyle\sum_{j=1}^{m} x_{ij} = 1 & & i \in 1, \ldots, n & (1.1) \\
& \displaystyle\sum_{i=1}^{n} w_i x_{ij} \leqslant C y_j & & j \in 1, \ldots, n & (1.2) \\
& x_{ij} \in \{0,1\} & & i \in 1, \ldots, n \ , \ j \in 1, \ldots, m & (1.3) \\
& y_j \in \{0,1\} & & j \in 1, \ldots, m & (1.4)
\end{array}\right]
$$

Par défaut, nous prenons $m = n$ ce qui assure l'obtention d'une solution réalisable triviale au problème (chaque objet affecté à un bin, ce qui conduit à une valeur de solution la plus défavorable). Nous verrons par la suite qu'il existe des heuristiques permettant d'obtenir une valeur pour m de bonne facture. Dans le modèle P1, (1.0) donne le nombre de bins utilisés, (1.1) assurent que chaque objet est assigné à un bin, (1.2) assurent que la capacité du bin n'est pas dépassée par les objets qui lui sont assignés, $(1.3 - 1.4)$ donnent les contraintes d'intégrités.

Modèle P2

$$\left[\begin{array}{lllll} \min \quad z = \displaystyle\sum_{j=1}^{m} y_j & & & & (2.0) \\ s.c. \quad \displaystyle\sum_{j=1}^{m} x_{ij} & = & 1 & i \in 1,\ldots,n & (2.1) \\ \displaystyle\sum_{i=1}^{n} w_i x_{ij} & \leqslant & C & j \in 1,\ldots,n & (2.2) \\ x_{ij} & \leqslant & y_j & i \in 1,\ldots,n \ , \ j \in 1,\ldots,m & (2.3) \\ x_{ij} & \in & \{0,1\} & i \in 1,\ldots,n \ , \ j \in 1,\ldots,m & (2.4) \\ y_j & \in & \{0,1\} & j \in 1,\ldots,m & (2.5) \end{array}\right]$$

Dans ce modèle P2, $(2.0, 2.1, 2.4 - 2.5)$ ont respectivement la même signification que $(1.0, 1.1, 1.3 - 1.4)$ dans P1, (2.2) assurent que la capacité d'un bin n'est pas dépassée par les objets qui lui sont assignés, (2.3) assurent qu'un objet peut être rangé dans un bin si celui-ci est utilisé.

1.6 Relaxation linéaire

Intéressons nous maintenant à la relaxation linéaire de ce problème de P1 mais également de P2. On modifie donc les contraintes d'intégrités $(1.3 - 1.4)$ (respectivement $(2.4 - 2.5)$) pour $i = 1,\ldots,n$ et $j = 1,\ldots,m$:

$$\left\{\begin{array}{l} 0 \leq x_{ij} \leq 1 \\ 0 \leq y_j \ \leq 1 \end{array}\right.$$

On considère les objets dans le sens où ils sont énoncés dans l'exemple didactique et on suppose que les objets sont divisibles et qu'ils peuvent être scindés. Ainsi, on peut remplir les bins jusqu'à atteindre leur capacité. Ainsi nous obtenons pour les bins la solution présentée TABLE 5 pour le problème P1. Soit c_j les capacités résiduelles des bins définies telles que $c_j = C - \displaystyle\sum_{i=1}^{n} w_i x_{ij}$ avec $i \in 1,\ldots,n$ et $j \in 1,\ldots,m$.

Pour cette solution, on a $c_j = 0, \ \forall\, j \in 1,\ldots,4$. Tous les bins sont donc remplis au maximum de leur capacité et tous les objets sont placés dans des bins. On atteint donc la borne duale

objets $o_i \ \forall i \in 1, \ldots, 7$	o_1 o_2 $\frac{1}{3}o_3$	$\frac{2}{3}o_3$ o_4 $\frac{1}{4}o_5$	$\frac{3}{4}o_5$ $\frac{3}{4}o_6$	$\frac{1}{4}o_6$ o_7
bins $b_j \ \forall j \in 1, \ldots, 4$	b_1	b_2	b_3	b_4

TABLE 5 – Solution optimale pour le problème P1

et la solution est optimale pour le modèle P1 relâché. En reprenant les notations définies dans la section 1.5, la solution s'écrit :

$x_{1,1} = 1$ $x_{2,1} = 1$ $x_{3,1} = \frac{1}{3}$	$x_{3,2} = \frac{2}{3}$ $x_{4,2} = 1$ $x_{5,2} = \frac{1}{4}$	$x_{5,3} = \frac{3}{4}$ $x_{6,3} = \frac{3}{4}$	$x_{6,4} = \frac{1}{4}$ $x_{7,4} = 1$	et $x_{i,j} = 0$ sinon
$y_1 = 1$	$y_2 = 1$	$y_3 = 1$	$y_4 = 1$	$y_5 = \cdots = y_m = 0$

Pour le modèle P1 relaxé, on obtient $z = \sum_{j=1}^{m} y_j = 4$ la valeur optimale. En reprenant les valeurs des x_{ij} correspondant à la solution obtenue relâchée ci-dessus (i.e. la solution obtenue en prenant les objets dans l'ordre donné par l'énoncé, objets pouvant être scindés), nous pouvons déduire les valeurs de y_j dans le cas du modèle P2.

La contrainte (2.3) impose que les valeurs x_{ij} soient inférieurs ou égales aux valeurs y_j. Le sens d'optimisation implique que les valeurs y_j soient aussi petites que possible. Étant donné que l'on déduit les valeurs y_j à partir des valeurs x_{ij} et que l'on minimise les valeurs y_j, on sait que la plus grande valeur y_j sera au moins égale (ou plus grande) à la plus petite valeur x_{ij} autre que 0. Ici, $\frac{1}{4}$ est la plus petite valeur parmi tous les x_{ij} donc on aura $y_j \geqslant \frac{1}{4}$ pour tout $j \in 1, \ldots, 4$. Ainsi, on obtient $z = \sum_{j=1}^{m} y_j = 1$ la valeur optimale pour le modèle P2 relâché.

On observe que bien que les modèles 1 et 2 sont équivalents en entier, ils ne le sont pas en continu et le modèle P2 est plus fort que le modèle P1.

2 Instances de BB1D et environnement de test

2.1 Présentation des instances

Pour conduire nos expérimentation numérique nous utilisons des collections d'instances numériques issues de la OR-library pour le bin packing (e.g. la librairie sur le bin packing de l'Università di Bologna sur https://site.unibo.fr/).

Pour résoudre le BB1D nous avons implémenter l'algorithme de branch-and-bound (BnB) présenté par Eilon et Christofides [1] et nous l'avons testé sur une collection d'instances présentée par E. Falkenauer [2] nommée `binpack1`. Le fichier correspondant à la collection est découpé comme suit :

- Le nombre d'instances (de problèmes) P
- Pour chaque instance ($p \in 1, \ldots, P$) :
  - ◇ l'identification du problème
  - ◇ la capacité des bins, le nombre d'objets n et le nombre de bins dans la meilleure solution connue (cette donnée n'est pas conservée pour la résolution du problème)
  - ◇ pour chaque objet i, l'espace w_i qu'il occupe (avec $i \in 1, \ldots, n$)

La collection d'instances `binpack1` comprend 20 instances dans lesquelles on considère des bins de capacité $C = 150$ et des 120 objets ($n = 120$) dont l'espace occupé a été généré aléatoirement avec une distribution uniforme. D'autres collections d'instances (`binpack2`, `binpack3`, ..., `binpack8`) n'ont pas été considérée car trop difficile.
Selon les résultats des expérimentations de Eilon et Christofides, l'algorithme BnB que nous avons implémenté ne peut résoudre que des instances de petites tailles en un temps raisonnable. De ce fait, nous avons utilisé des instances plus petites présentées par A. Scholl, R. Klein, et C. Jürgens [3]. Certaines de ces instances ont été regroupées par nous dans des fichiers du même format que la collection `binpack1`. Toutes les instances avec les mêmes caractéristiques ont été regroupées dans un même fichier (e.g. `N1C1W1_A`, `N1C1W1_B`, ..., `N1C1W1_T` ont été regroupées dans la collection `N1C1W1`). Les instances choisies proviennent tous de l'ensemble "Scholl 1", un ensemble de 720 instances uniformément distribuées avec n entre 50 et 500. La capacité C des bins de ces instances est entre 100 et 150. Le nom des collections est définie comme suit : `Nx` est le nombre d'objets (e.g. `N1` = 50, `N2` = 100, `N3` = 200), `Cx` est la capacité des bins (e.g. `C1` = 100, `C2` = 120, `C3` = `150`), et `Wx` correspond à la distribution utilisée pour générer les poids des objets (e.g. `W1` = $[1, 100]$, `W2` = $[20, 100]$).

2.2 Environnement de test

L'ensemble des algorithmes que nous présenterons ci-après sont implémentés en Julia (version 1.8.1). Dés qu'une résolution de problème est nécessaire l'environnement JuMP/ MOI est utilisé avec le solveur GLPK (en utilisant des ré-optimisations tant que cela est possible). L'ensemble des tests ont été réalisé sur une machine équipée d'un processeur Intel® Core™ i7-9750H de 9ème génération (6 coeurs, 12 threads, 2.6 GHz de base / 4.50 GHz maximum) et avec un temps limite de 45 secondes ("timeout").

3 Branch-and-bound pour le BB1D

3.1 Algorithmes pour calculer la borne primale

Il existe dans la littérature un très grand nombre d'algorithmes approchés pour calculer une solution admissible au BB1D. Tous nos objets étant connus a priori, un pré-traitement peut leur être appliqué. Supposons que les objets sont triés par espace occupé décroissant ($w_i \geqslant w_{i+1}$, $i \in 1, \ldots, n-1$). Les heuristiques gloutonnes suivantes permettent d'obtenir une solution réalisable et une borne primale :

- *next-fit decreasing* (NFD) :
  range l'objet suivant dans le bin actuel si possible, ou dans un nouveau bin lequel devient le bin courant sinon ;
- *first-fit decreasing* (FFD) :
  range l'objet suivant dans le bin utilisé de plus petit indice pouvant l'accueillir, ou dans un nouveau bin si aucun bin utilisé ne peut accueillir l'objet ;
- *best-fit decreasing* (BFD) :
  range l'objet suivant dans un bin utilisé qui peut l'accueillir, de manière à laisser le plus petit espace résiduel possible, ou dans un nouveau bin si aucun bin utilisé ne peut accueillir l'objet ;

Les résultats que ces heuristiques obtiennent sur les collections `binpack1` et `N1C1W1` sont donnés Table 6 et Table 7. Les heuristiques nécessitent toutes environ le même temps CPUt donc il n'a pas été rapporté sur ces tables. Nous observons sur les tables que l'heuristique NFD est celle qui obtient des bornes primales de moins bonne qualité comparé aux autres. D'un autre côté, FFD et BFD ont toutes les deux les mêmes résultats sur toutes les instances testées.

3.2 Implémentations d'un branch-and-bound pour le BB1D

3.2.1 Première version du BnB

Nous avons reproduit l'algorithme proposé dans la littérature par Eilon et Christofides [1] en instrumentant notre solveur pour en mesurer l'activité (e.g. nombre de noeuds visités, temps consommé) ou l'arrêter (e.g. délai d'exécution à ne pas dépasser). Cette première version est dénommée *BnB1*. Tout sous-problème dérivé d'un problème est résolu par ré-optimisation du problème initial. Enfin, la borne inférieure que nous utilisons est celle discutée en section 1.4. Plus explicitement, cette borne s'écrit

$$L_1 = \left\lceil \sum_{j=1}^{n} \frac{w_j}{c} \right\rceil$$

Sur les tables 8 et 9 figurent la valeur **z** obtenue, le **nombre de noeuds visités**, le **nombre de solutions réalisables trouvées** et le **temps CPUt** pour l'instance considérée obtenus par *BnB1* et ses améliorations (présentées ci-après) *BnB2* et *BnB3*.

3.2.2 Deuxième version du BnB

Martello et Toth [4] ont proposé une borne L_2 dominant la borne L_1. Soit $\alpha \in \mathbb{N}$ tel que $0 \leqslant \alpha \leqslant C/2$, notons

$$\begin{aligned} J_1 &= \{i \in 1, \ldots, n : w_i > C - \alpha\}, \\ J_2 &= \{i \in 1, \ldots, n : C - \alpha \geqslant w_i > C/2\}, \\ J_3 &= \{i \in 1, \ldots, n : C/2 \geqslant w_i \geqslant \alpha\}; \end{aligned}$$

alors

$$L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{i \in J_3} w_i - (|J_2|C - \sum_{i \in J_2} w_i)}{C} \right\rceil\right)$$

est une borne inférieure sur le nombre de bins nécessaires.

On définit alors la borne L_2 comme suit

$$L_2 = \max\{L(\alpha) : 0 \leqslant \alpha \leqslant C/2,\ \alpha \in \mathbb{N}\}$$

Dans cette deuxième version du BnB, que nous appelerons *BnB2*, nous utiliserons L_2 au lieu de L_1.

3.2.3 Troisième version du BnB

Toujours dans la perspective d'améliorer l'efficacité de l'algorithme, dans cette troisième du BnB nous ajoutons au fur et à mesure de la résolution des coupes définissant des inégalités valides. Celles-ci sont ajoutées au problème d'optimisation en vue de resserer la relaxation linéaire.

Dans le modèle P1, la coupe

$$x_{5,j} + x_{6,j} + x_{7,j} \leqslant 1 \quad \forall\, j \in 1, \ldots, m$$

est une inégalité valide pour l'instance didactique. En effet, lorsque l'on construit une solution en scindant les objets (problème relâché) il est possible d'obtenir une solution admissible pour le problème en entiers en reconstruisant les objets scindés et en les plaçant dans des bins. Cependant, en reconstruisant les objets, il se peut qu'il y ai toujours des objets qui, lorsqu'ils ne sont pas scindés, ne peuvent être ensemble. C'est le cas par exemple des objets 5, 6 et 7 dans l'instance didactique ($w_5 + w_6 + w_7 = 13 > 6 = C$).

La contrainte de sac-à-dos présente dans la modélisation a fait l'objet d'attention particulière dans la littérature et a conduit à la définition de *cover cuts*. Kaparis K. et Letchford A. [5] présentent dans leur papier plusieurs inégalités de couverture pour les problèmes de sac-à-dos et dérivés. Nous définirons ici quelques inégalités de couverture avant de sélectionner un type d'inégalités de couverture pour améliorer notre BnB.

Soit une contrainte de sac-à-dos

$$\sum_{j=1}^{n} a_j x_j \leqslant b$$

Définition 1

Un ensemble $C \subseteq N = \{1, \ldots, n\}$ est une *couverture* (*cover*) si $\sum_{j \in C} a_j x_j > b$

Définition 2

Une *couverture* est *minimale* (*minimal cover*) si $C \backslash \{j\}$ n'est pas une couverture pour n'importe quel $j \in C$.

Définition 3

Si $C \subseteq N = \{1, \ldots, n\}$ est une couverture alors $\sum_{j \in C} x_j \leqslant |C| - 1$ est une *inégalité valide*.

Définition 4

Si $C \subseteq N = \{1, \ldots, n\}$ est une couverture pour $\sum_{j=1}^{n} a_j x_j \leqslant b$ alors $\sum_{j \in E(C)} x_j \leqslant |C| - 1$ est une inégalité valide, où E(C) est une *extended cover* définie par

$$E(C) = C \cup \{j \,|\, \forall\, i \in C : a_j \geqslant a_i\}$$

On parle de *stronger cover cut*.

Dans notre algorithme nous avons implémenté les inégalités de couverture minimale (*BnB3*).

Pour calculer une inégalité de couverture, nous considérons un bin et nous vérifions que la somme des poids des objets (dans l'ordre décroissant des poids) présents dans le bin ne dépasse pas la capacité du bin. Cette somme est calculée au fur et à mesure et dés qu'elle dépasse la capacité du bin alors les indices des variables considérées jusque là forment une couverture. De plus, cette couverture est minimale car en soustrayant le poids de n'importe quel objet dans la couverture à la somme obtenue, la somme ne dépasse plus la capacité du bin (et donc l'ensemble d'indices correspondant n'est pas une couverture).

3.3 Comparaison des différentes versions de BnB

La TABLE 8 nous montre que les algorithmes implémentés ont beaucoup de difficulté à fournir des solutions pour des instances volumineuses (les instances de la collection `binpack1`) en un temps raisonnable (ici le temps accordé à chaque instance est de 45 secondes et le nombre de bins obtenus correspond au nombre de bins fournit par l'heuristique BFD lorsqu'aucune solution réalisable n'est trouvée). Cependant, nous pouvons observé que la troisième version *BnB3* parcours beaucoup moins de noeuds que les deux autres versions. Sur des plus petites instances (voir TABLE 9), nous remarquons que *BnB2* visite souvent moins de noeuds que *BnB1* mais généralement, plus la taille des instances est grande moins la différence est notable (e.g. sur l'instance `N1C1W1_C` *BnB1* visite 10165 noeuds contre 101 pour *BnB2* mais sur l'instance `N1C1W1_D` le nombre de noeuds visités est presque équivalent pour ces deux premières versions du BnB). Étant donné qu'ils visitent souvent moins de

noeuds que *BnB1*, les versions *BnB2* et *BnB3* ont des meilleurs temps CPUt. En contrepartie, *BnB2* et *BnB3* trouvent systématiquement moins ou autant de solutions que *BnB1* bien que cela ne change pas le résultat final obtenu.

Un désavantage notable de *BnB3* c'est que les coupes ajoutées peuvent parfois élaguer des branches intéressantes. Ainsi, *BnB3* peut ne pas fournir de solutions réalisables alors que les autres versions en fournisse (i.e. les instances `didactic`, `N1C1W1_F` et `N1C1W1_O`). Enfin, *BnB3* est globalement plus efficace que *BnB2* sur les instances volumineuses, trouvant des solutions réalisables en moins de temps que *BnB2* sur les instances `N1C1W1_I`, `N1C1W1_P`, `N1C1W1_R` et `N1C1W1_T`. Sur les petites instances, *BnB2* et *BnB3* visitent généralement le même nombre de noeuds et, étant donné que *BnB3* effectue plus d'opérations à cause du calcul des coupes, *BnB2* est plus rapide que *BnB3*.

Références

[1] Samuel Eilon et Nicos Christofides. « The loading problem ». In : *Management Science* 17.5 (1971), p. 259-268.

[2] Emanuel Falkenauer. « A hybrid grouping genetic algorithm for bin packing ». In : *Journal of heuristics* 2.1 (1996), p. 5-30.

[3] Armin Scholl, Robert Klein et Christian Jürgens. « Bison : A fast hybrid procedure for exactly solving the one-dimensional bin packing problem ». In : *Computers & Operations Research* 24.7 (1997), p. 627-645.

[4] Silvano Martello et Paolo Toth. « Lower bounds and reduction procedures for the bin packing problem ». In : *Discrete applied mathematics* 28.1 (1990), p. 59-70.

[5] Konstantinos Kaparis et Adam N Letchford. « Cover inequalities ». In : *Wiley Encyclopedia of Operations Research and Management Science* (2010).

	m			
	NFD	FFD	BFD	
etude	3	3	3	etude
didactic	5	5	5	didactic
U120_00	67	49	49	U120_00
U120_01	67	49	49	U120_01
U120_02	62	47	47	U120_02
U120_03	69	50	50	U120_03
U120_04	69	50	50	U120_04
U120_05	68	49	49	U120_05
U120_06	68	49	49	U120_06
U120_07	67	50	50	U120_07
U120_08	71	51	51	U120_08
U120_09	64	47	47	U120_09
U120_10	75	52	52	U120_10
U120_11	71	50	50	U120_11
U120_12	68	49	49	U120_12
U120_13	68	49	49	U120_13
U120_14	68	50	50	U120_14
U120_15	67	49	49	U120_15
U120_16	73	52	52	U120_16
U120_17	75	53	53	U120_17
U120_18	66	50	50	U120_18
U120_19	69	50	50	U120_19
	NFD	FFD	BFD	
	m			

TABLE 6 – Comparaison des bornes obtenus avec les différentes heuristiques sur la collection d'instances `binpack1`.

	m			
	NFD	FFD	BFD	
etude	3	3	3	etude
didactic	5	5	5	didactic
N1C1W1_A	30	25	25	N1C1W1_A
N1C1W1_B	35	31	31	N1C1W1_B
N1C1W1_C	26	21	21	N1C1W1_C
N1C1W1_D	32	28	28	N1C1W1_D
N1C1W1_E	31	26	26	N1C1W1_E
N1C1W1_F	32	27	27	N1C1W1_F
N1C1W1_G	30	25	25	N1C1W1_G
N1C1W1_H	35	31	31	N1C1W1_H
N1C1W1_I	31	25	25	N1C1W1_I
N1C1W1_J	32	26	26	N1C1W1_J
N1C1W1_K	31	26	26	N1C1W1_K
N1C1W1_L	38	33	33	N1C1W1_L
N1C1W1_M	34	30	30	N1C1W1_M
N1C1W1_N	30	26	26	N1C1W1_N
N1C1W1_O	37	32	32	N1C1W1_O
N1C1W1_P	32	26	26	N1C1W1_P
N1C1W1_Q	35	28	28	N1C1W1_Q
N1C1W1_R	31	25	25	N1C1W1_R
N1C1W1_S	33	28	28	N1C1W1_S
N1C1W1_T	34	28	28	N1C1W1_T
	NFD	FFD	BFD	
	m			

TABLE 7 – Comparaison des bornes obtenus avec les différentes heuristiques sur la collection d'instances `N1C1W1`.

	z			# noeuds visités			# solutions trouvées			CPUt (s)			
	BnB1	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	
etude	3	3	3	59	13	13	2	2	2	0.28	0.28	0.37	etude
didactic	5	5	5	4511	4483	93	2	1	0	0.98	0.97	0.32	didactic
U120_00	49	49	49	808	819	54	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_00
U120_01	49	49	49	803	803	57	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_01
U120_02	47	47	47	802	790	42	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_02
U120_03	50	50	50	629	584	52	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_03
U120_04	50	50	50	597	593	54	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_04
U120_05	49	49	49	807	769	54	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_05
U120_06	49	49	49	778	760	57	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_06
U120_07	50	50	50	383	363	42	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_07
U120_08	51	51	51	860	791	53	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_08
U120_09	47	47	47	929	894	44	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_09
U120_10	52	52	52	756	737	55	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_10
U120_11	50	50	50	665	625	57	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_11
U120_12	49	49	49	467	465	52	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_12
U120_13	49	49	49	769	730	47	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_13
U120_14	50	50	50	757	743	53	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_14
U120_15	49	49	49	816	813	54	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_15
U120_16	52	52	52	752	766	59	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_16
U120_17	53	53	53	772	782	53	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_17
U120_18	50	50	50	607	607	48	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_18
U120_19	51	51	51	759	758	54	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	U120_19
	BnB1	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	
	z			# noeuds visités			# solutions trouvées			CPUt (s)			

TABLE 8 – Comparaison des résultats obtenus avec la version BnB1 et la version BnB2 sur la collection d'instances `binpack1`.

	z			**# noeuds visités**			**# solutions trouvées**			**CPUt (s)**			
	BnB1	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	
etude	3	3	3	59	13	13	2	2	2	0.28	0.28	0.37	etude
didactic	5	5	5	4511	4483	93	2	1	0	0.98	0.97	0.37	didactic
N1C1W1_A	25	25	25	8659	101	101	1	1	1	TIMEOUT	2.87	3.67	N1C1W1_A
N1C1W1_B	31	31	31	5864	5817	347	1	1	1	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_B
N1C1W1_C	21	21	21	10165	101	101	1	1	1	TIMEOUT	1.68	2.13	N1C1W1_C
N1C1W1_D	28	28	28	6912	6925	515	1	1	1	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_D
N1C1W1_E	26	26	26	8506	99	99	18	1	1	TIMEOUT	2.58	3.28	N1C1W1_E
N1C1W1_F	27	27	27	7630	103	332	1	1	0	TIMEOUT	3.05	TIMEOUT	N1C1W1_F
N1C1W1_G	25	25	25	8571	99	99	1	1	1	TIMEOUT	2.67	3.38	N1C1W1_G
N1C1W1_H	31	31	31	7121	6889	357	1	1	1	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_H
N1C1W1_I	25	25	25	9207	8726	167	69	69	1	TIMEOUT	TIMEOUT	7.27	N1C1W1_I
N1C1W1_J	26	26	26	8597	101	101	2	2	2	TIMEOUT	2.74	3.5	N1C1W1_J
N1C1W1_K	11	26	26	8006	97	97	11	1	1	TIMEOUT	2.6	3.4	N1C1W1_K
N1C1W1_L	33	33	33	6024	5938	298	1	1	1	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_L
N1C1W1_M	30	30	30	7323	7147	372	6	6	2	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_M
N1C1W1_N	26	26	26	8751	8637	337	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_N
N1C1W1_O	32	32	32	6211	5981	259	1	1	0	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_O
N1C1W1_P	26	26	26	7950	7725	617	2	2	2	TIMEOUT	TIMEOUT	34.45	N1C1W1_P
N1C1W1_Q	28	28	28	7253	7159	583	1	1	1	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_Q
N1C1W1_R	25	25	25	8933	101	101	2	2	2	TIMEOUT	2.5	2.45	N1C1W1_R
N1C1W1_S	28	28	28	8433	8285	396	0	0	0	TIMEOUT	TIMEOUT	TIMEOUT	N1C1W1_S
N1C1W1_T	28	28	28	8021	101	101	2	2	2	TIMEOUT	3.2	3.1	N1C1W1_T
	BnB1	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	*BnB1*	*BnB2*	*BnB3*	
	z			**# noeuds visités**			**# solutions trouvées**			**CPUt (s)**			

TABLE 9 – Comparaison des résultats obtenus avec la version BnB1 et la version BnB2 sur la collection d'instances N1C1W1.