Rapport du TP OPTIMISATION Pt.1: Methodes Exactes

BACHI Yasmine (CdE) SAADI Fatma Zohra Khaoula
NOUALI Sarah MOUSSAOUI Meroua
MIHOUBI Lamia Zohra

30 avril 2020

Table des matières

Introduction

Le problème du bin packing, dans lequel un ensemble d'objets de différents poids doit être rangé dans un nombre minimum de boîtes identiques de capacité C est un problème NP-difficile, c'est à dire qu'il n y a aucune chance de trouver une méthode de résolution qui fournit la solution exacte en un temps polynomiale, sauf si l'égalité NP=P est prouvée. Durant le dernier siècle, divers efforts ont été consacrés pour étudier ce problème, dans le but de trouver des algorithmes heuristiques rapides pour fournir de bonnes solutions approximatives. Dans ce projet, nous allons mettre en place un plateforme de résolution du problème du Bin Packing. pour cela , nous implémenterons 4 types de méthodes :

1. *méthodes exactes :* fournissant la solution optimale, mais qui sont très limités par la taille du problème.
2. *heuristiques :* qui sont des méthodes approchées spécifiques au problème.
3. *métaheuristiques :* qui sont des méthodes approchées génériques.
4. *hybridation d'une métaheuristique avec une recherche locale :* qui est notre contribution principale dans la résolution de ce problème.

Nous commencerons par la présentation du problème, sa formulation mathématique, et une étude des méthodes de résolutions existantes dans la littérature.[Etat de l'Art]. Ensuite, nous présenterons la conception détaillée de chaque méthode implémentée, ainsi que les résultats des test de ces méthodes effectués sur des benchmark connus.[Conception & Tests] On distingue 2 types de tests :

1. *les tests empiriques :* dont le but de trouver la meilleure configuration des paramètres de nos méthodes implémentées.
2. *les tests comparatifs :* où on doit comparer les résultats obtenus des méthodes implémentées et sélectionner la meilleure méthode de résolution pour chaque instances. la comparaison se fait en terme de qualité de la solution et du temps d'exécution.

Chapitre 1

Présentation du Problème de Bin Packing (BPP)

1.1 Domaines d'Application :

Le BPP a de nombreuses applications dans le domaine industriel, informatique, etc. Parmi lesquelles on trouve :

- Chargement de conteneurs.
- Placement des données sur plusieurs disques.
- Planification des travaux.
- Emballage de publicités dans des stations de radio / télévision de longueur fixe.
- Stockage d'une grande collection de musique sur des cassettes / CD, etc.

1.2 Formulation Mathématique

Etant donné m boites de capacité C et n articles de volume v_i chacun. Soient :

$$x_{ij} = \begin{cases} 1 & \text{article } j \text{ rangé dans la boîte } i \\ 0 & \text{sinon} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{boîte } i \text{ utilisée} \\ 0 & \text{sinon} \end{cases}$$

La formulation du problème donne ainsi le programme linéaire suivant

$$(PN)\begin{cases} Z(min) = \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_{ij} = 1 \\ \sum_{j=1}^{n} v_j x_{ij} \leq C y_i \\ y_i \in \{0,1\} \\ x_{ij} \in \{0,1\} \end{cases}$$

La première contrainte signifie qu'un article j ne peut être placé qu'en une seule boite La deuxième fait qu'on ne dépasse pas la taille d'une boite lors du rangement

Chapitre 2

Etat de l'Art

Après une étude ciblée des travaux existants sur le problème du Bin Packing, nous avons abouti à une synthèse des méthodes de résolution les plus connues –dans chacune des trois catégories : méthodes exactes, et méthodes approchées : heuristiques et métaheuristiques– que nous allons exposer dans ce qui suit.

2.1 Méthodes Exactes

Les méthodes exactes permettent d'avoir des solutions optimales, cependant le temps de calcul peut être très long pour certaines instances du problème. Il n'existe pas un grand nombre de méthodes exactes pour résoudre le problème du Bin Packing , nous allons présenter dans ce qui suit la méthode MTP (basée sur le Branch and Bound) et une méthode de programmation dynamique DP-flow.

2.1.1 Branche and Bound

Cette méthode a été utilisée pour la première fois dans les années cinquante pour résoudre qui a été modélisé par un programme linéaire en nombres entiers. Afin de rendre le processus de résolution plus rapide, cet algorithme utilise une borne inférieure. Plusieurs techniques ont été proposés pour obtenir cette dernière.

Borne inférieure évidente : Soient C la capacité des boîtes utilisés, A l'ensemble des articles a_i de volumes v_i de l'instance I.

$$BI(I) = \frac{\sum_{i=1}^{n} v_i}{C}$$

Borne de Martello and Toth L2 : Soit α un entier tels que :

$$0 \leq \alpha \leq C/2$$

On definie des classes d'articles suivantes :

$$C_1 = \{a_i, \quad C - \alpha < v_i\}$$
$$C_2 = \{a_i, \quad C/2 < v_i \leq C - \alpha\}$$
$$C_3 = \{a_i, \quad \alpha < v_i \leq C/2\}$$

$BI(I)$ est donnée par la formule suivante :

$$BI(I) = max\{L(\alpha), \quad 0 \leq \alpha \leq C/2\}$$

Avec

$$L(\alpha) = |C_1| + |C_2| + max(0, \left\lceil \frac{\sum_{j \in C_3} v_j - (|C_2| * C - \sum_{j \in C_2} v_j)}{C} \right\rceil)$$

cette borne est calculé en un temps $o(nlogn)$.

Borne inférieure L_3 : Une autre technique a été utilisé dans l'algorithme de résolution MTP pour déterminer une borne inférieure. Soient n_1 le nombre de boîtes obtenus après la première application de la technique de réduction MTP qui consiste à réduire l'instance du problème en rangeant l'article le plus petit, soit Ir^1 l'instance résiduelle de l'instance I après cette première application ie l'ensemble des articles restants après l'opération de réduction

$$L'_1 = n_1 + L_2(Ir^1) \geq L_2(I)$$

On refait ce processus jusqu'à ce que l'instance résiduelle soit vide (i.e. : tous les articles ont été rangés). A l'itération k, on obtiendra :

$$L'_K = \sum_{i=1}^{k} n_i + L_2(Ir^k)$$

La borne inférieure L_3 est obtenue en appliquant la formule suivante , par la suite :

$$L_3 = max\{L'_1, L'_2, \cdots, L'_{kmax}\}$$

Un des algorithmes proposés pour cette méthode est l'algorithme MTP :

L'algorithme MTP (Martello and Toth Procédure) :

Le meilleur algorithme existant pour trouver la solution optimale du problème Bin Packing est celui proposée par *Martello et Toth (Martello & Toth 1990a ; 1990b)* le principe est le suivant : Les articles sont initialement triés selon des poids décroissants. À chaque nœud de décision, le premier élément libre est attribué, à son tour, aux boîtes existantes qui peuvent le contenir (on parcourt les boîtes par ordre de création) et à une nouvelle boîte. À chaque nœud de l'arbre de recherche

a. Une borne inférieure L_3 de la solution restante est calculée et utilisée pour élaguer le nœud de l'espace de recherche et réduire le problème actuel.
   - Si la borne inférieure du nœud actuel est supérieure à la borne inférieure du problème d'origine (nœud racine), le nœud est supprimé. Sinon (b)

b. Des algorithmes approximatifs FFD, BFD et WFD (qu'on présentera dans la partie Méthodes approximatives) sont appliqués au problème actuel, et chacune des solutions approximatives obtenues est comparée à la borne inférieure L_3.
   - Si le nombre de boîtes utilisées par l'une des solutions approximatives est égal à la borne inférieure du nœud actuel, aucune autre recherche n'est effectuée sous ce nœud.
   - Si le nombre de boîtes utilisées dans une solution approximative est égal à la borne inférieure L3 du problème d'origine (nœud racine), l'algorithme se termine, renvoyant cette solution comme optimale.

De plus, La principale source d'efficacité de l'algorithme de Martello et Toth est une méthode pour réduire la taille des sous-problèmes restants, appelée **critère de dominance**.

2.1.2 Programmation Dynamique

La programmation dynamique est une méthode algorithmique pour résoudre des problèmes d'optimisation. Le concept a été introduit au début des années 1950 par Richard Bellman. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires. Elle s'appuie sur *le principe d'optimalité de Bellman* : une solution optimale d'un problème s'obtient en combinant des solutions optimales à des sous-problèmes. Un modèle pseudo-polynomial simple, pour résoudre le problème bin-packing,

est obtenu en associant des variables aux décisions prises dans une table de programmation dynamique (DP) classique.

DP-flow :

Dans le modèle BPP proposé par *Cambazard et O'Sullivan*, connu sous le nom de DP-flow, les états DP sont représentés par un graphique dans lequel un chemin qui commence à partir d'un nœud initial et se termine à un nœud terminal représente un remplissage possible d'une boîte. Notons $(j, d)(j = 0, \cdots, n\, et\, d = 0, \cdots, c)$ un état DP où les articles de 0 à j ont déjà
été étudié (les décisions de placer les articles de 0 à j dans la boîte ou non ont déjà été reprises) et entraînent un remplissage partiel de la boîte de d unités. Notons également par $((j, d), (j + 1, e))$ un arc reliant les états (j, d) et $(j+1, e)$. Un tel arc exprime la décision d'emballer ou non l'article $j+1$ à partir de l'état actuel (j, d) : l'état atteint par l'arc est $(j + 1, d + w_j + 1)$ si l'article $j+1$ est emballé, et $(j+1, d)$ sinon. Soit A l'ensemble de tous les arcs. Comme un remplissage réalisable d'une boîte est représenté par un chemin qui commence à partir du nœud $(0, 0)$ et se termine au nœud $(n + 1, c)$, le BPP consiste à sélectionner le nombre minimum des chemins qui contiennent tous les éléments.
Pour cette instance BPP, une solution optimale est produite par les deux chemins mis en évidence dans la figure ci-dessous, à savoir
$[(0, 0), (1, 4), (2, 4), (3, 7), (4, 7), (5, 9), (6, 9), (7, 9)]$
et $[(0, 0), (1, 0), (2, 4), (3, 4), (4, 7), (5, 7), (6, 9), (7, 9)]$.

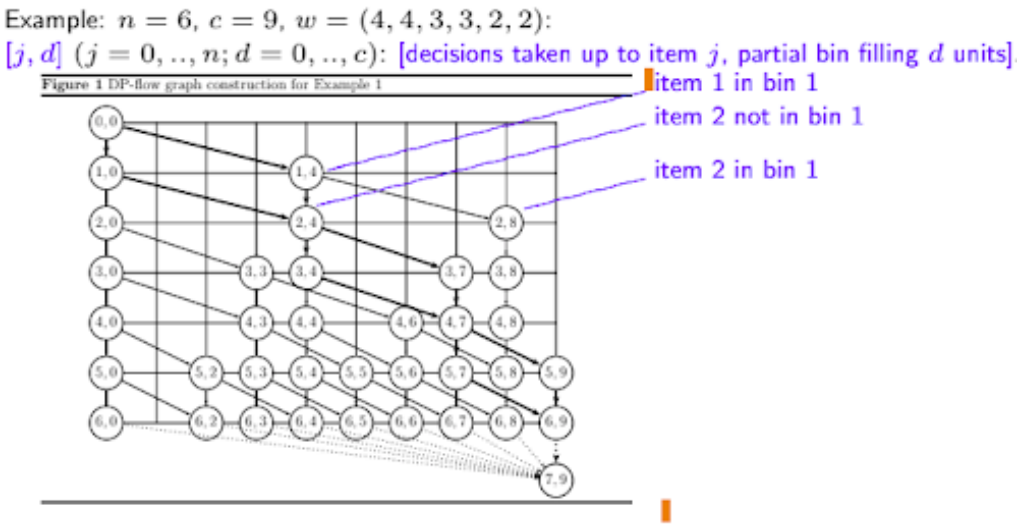


Figure 1 DP-flow graph construction for Example 1

2.2 Heuristiques

Les méthodes approchées sont classées en plusieurs catégories :

2.2.1 Algorithmes ON-LINE :

Ces algorithmes considèrent l'hypothèse que les articles arrivent un à la fois en un ordre connu, chaque article doit être rangé avant de passer à l'article suivant.

2.2.1.1 Next Fit (NF) :

Lors du rangement de l'article a, NF vérifie s'il tient dans la même boîte que le dernier article (Soit la boîte du dernier article rangé Bj). Si c'est le cas, il place l'article dans la boîte Bj, laissant cette boîte ouverte. Sinon, il ferme la boîte Bj et place l'article a dans une nouvelle boîte Bj+1, qui devient maintenant la boîte ouverte.
Remarque :
Il existe des variants pour cet algorithme, permettant d'améliorer ses performances :

— **Next-KFIT (NFk) :** autorise k boîte ouvertes à la fois, c'est à dire qu'il vérifie dans les K boîtes ouvertes s'il y a assez d'espace pour ranger l'article a, sinon il ouvre une nouvelle boîte. Si K=1 on retombe sur l'algorithme NF.

2.2.1.2 First Fit (FF) :

Lors du rangement de l'article a, NF vérifie s'il tient dans la même boîte que le dernier article (Soit la boîte du dernier article rangé Bj). Si c'est le cas, il place l'article dans la boîte Bj, laissant cette boîte ouverte. Sinon, il ferme la boîte Bj et place l'article a dans une nouvelle boîte Bj+1, qui devient maintenant la boîte ouverte.

2.2.1.3 Best Fit (BF) :

L'article a est rangé dans une des boîtes ouvertes de sorte que le plus petit espace vide soit laissé. Si l'article ne tient dans aucune boîte existante, il sera placé dans une nouvelle boîte.
Remarque :
Il existe des variants pour cet algorithme, permettant d'améliorer ses performances :

— **K-Bounded Best Fit (BBFk) :** utilise le même principe que Best Fit, sauf qu'il restreint le nombre de boîtes ouvertes à k boîtes. ie : l'article a est rangé dans une des k-boîtes ouvertes de sorte que le plus petit espace vite soit laissé.

2.2.1.4 Worst Fit (WF) :

L'article a est rangé dans la boîte avec le plus grand espace vide, si cette dernière ne peut pas contenir l'article, il sera placé dans une nouvelle boîte.
Remarque :
Il existe des variants pour cet algorithme, permettant d'améliorer ses performances :

— **Almost Worst Fit (AWF) :** l'article a est rangé dans la 2ème boîte la plus vide, jusqu'à ce qu'il reste qu'une seule boîte qui peut contenir l'article. Dans ce cas il est placé dans cette boîte. Si l'article ne tient dans aucune boîte existante, il sera placé dans une nouvelle boîte.

2.2.1.5 Harmonic K (Hk) :

Cet algorithme est basé sur une partition de l'intervalle $[0, 1]$ en K sous-intervalles I_k, où $I_k =]1k+1; 1k]$, à chacun de ces sous-intervalles correspond une seule boîte ouverte, et seuls les articles appartenant à ce sous-intervalle (i.e. : $v_i/C \in I_k$ avec v_i le volume d'un article) sont regroupés dans cette boîte. Si un nouvel article arrive et ne rentre pas dans sa boîte ouverte correspondante, la boîte est fermée et une nouvelle boîte est ouverte.
Remarque :
Il existe des variants pour cet algorithme, permettant d'améliorer ses performances :

— **Simplified HarmonicK (SHk) :** se base sur une structure d'intervalle qui est plus compliquée.

2.2.1.6 Autres algorithmes :

— **ABFk :** utilise le principe de rangement du Best Fit avec le principe de fermeture des boîtes du First Fit.

— **AFBk :** utilise le principe de rangement du First Fit avec le principe de fermeture des boîtes du Best Fit.

Remarque : Dans le cas où $K = 1$, ces algorithmes sont équivalent à l'algorithme Next Fit.

2.3 Métaheuristiques

2.3.1 La recherche tabou :

Une méthode de résolution a été proposée par Fernandes Muritiba Et al.[2] ,qui prend en entrée une population initiale obtenue par une heuristique et applique un opérateur de croisement sur ces solutions .Chaque solution obtenue est améliorée par la suite en utilisant une recherche tabou, qui consiste à se déplacer dans un espace de recherche contenant des solutions réalisables partielles où certains articles ne sont pas affectés à des boîtes. L'amélioration consiste de passer d'une solution partielle de valeur K à une solution complète de la même valeur. La fonction objective utilisée est basée sur la somme pondérée des tailles des articles. Pour la diversification de l'espace de recherche, On utilise une procédure basée sur un opérateur de croisement.

2.3.2 L'algorithme ILWOA (Improved Lévy WOA) :

L'algorithme WOA (Whale Optimization Algorithm) est basé sur une méthode inspirée de la nature plus exactement d'une stratégie d'alimentation des baleines à bosse, où la recherche des proies représente l'exploration de l'espace de recherche et la libération des bulles représente l'exploitation. Une amélioration de cet algorithme a été proposée par Abdel-Basset, M., Manogaran, G., Abdel-Fatah, L. et al [2].Il s'agit de l'utilisation de fonctions logistiques et d'autres méthodes probabilistes pour assurer une convergence plus rapide

2.3.3 L'algorithme FFA (FireFly Algorithm) :

L'algorithme FireFly (FFA) est une métaheuristique génétique, inspirée par le comportement clignotant des lucioles. Le but principal du flash d'une luciole est d'agir comme un système de signal pour attirer d'autres lucioles. Il y a trois règles. La première règle, chaque luciole attire tous les autres lucioles avec des flashs plus faibles. Deuxièmement, l'attractivité est proportionnelle à leur luminosité qui est inversement proportionnelle à leurs distances. Pour deux lucioles clignotantes, la moins brillante se déplacera vers la plus brillante. L'attractivité est proportionnelle à la luminosité et ils diminuent tous les deux à mesure que leur distance augmente. S'il n'y en a pas plus brillant qu'une luciole particulière, il se déplacera de façon aléatoire. Enfin, aucune luciole ne peut attirer la luciole la plus brillante, cette dernière se déplace d'une façon aléatoire. Le FFA est construit par analogie, en appliquant ces trois règles sur une population de solutions initiale pour la faire

évoluer en une population contenant la solution approchée.

Chapitre 3

Conception

Dans ce chapitre,nous allons présenter la conception détaillée des méthodes exactes sur lesquelles notre choix d'implementation s'est porté :

1. Le branch and bound
2. Une version améliorée du branch and bound
3. La recherche exhaustive
4. La programmation dynamique

Dans le but de montrer l'applicabilité de ces méthodes, comparer leurs performances et montrer leurs limites, nous effectuerons des tests empiriques et comparatifs sur des benchmarks d'un coté, et sur des instances générées par notre propre générateur d'instances d'un autre coté.

3.1 Branch and bound

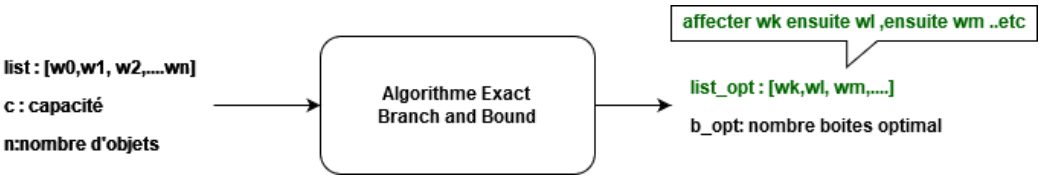
— L'algorithme Branch-and-Bound (B &B) que nous avons implémenté tente de ranger un objet à la fois en fonction de l'ordre initial des objets.

— Au niveau j de l'arbre, B &B crée un noeuds fils pour chaque boite ouverte et range l'objet j dans cette boîte si c'est possible. il crée aussi un noeuds supplémentaire qui représente l'ouverture d'une nouvelle boîte, et il range l'objet j dans cette boîte.

— En pratique, au niveau 1 de l'arbre l'objet 1 est rangé dans la boîte 1 , au niveau 2 l'objet 2 est rangé dans la boîte 1 ou dans une nouvelle boîte 2 ,...etc

— A chaque noeud, on résout un sous problème de taille (n-k) du bin packing, où les k premiers objets ont déjà été emballés.

— L'opération du rangement d'un objet i au niveau k consiste à permuter entre les éléments list(K) et list(i). On va avoir comme sortie une liste d'objets ordonnées selon l'ordre de rangement, il suffit ensuite de remplir les boîtes par les objets dans leur nouvel ordre pour générer la solution (l'emplacement de chaque objet dans les boîtes)



3.1.1 Pseudo-Code

Evaluation d'un noeud (Borne L1)

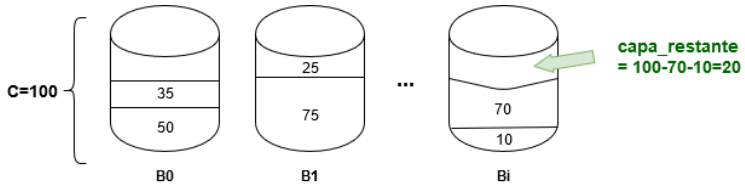
L'évaluation d'un nœud est calculée en sommant 2 parties, le nombre de boîtes déjà utilisées bcount et une estimation du nombre de boîtes qu'on va ouvrir encore pour contenir les objets restants. Cette estimation est obtenue en divisant la somme des poids restants sumwt sur la capacité d'une boîte. On soustrait de la somme des poids restants, l'espace vide restant dans la dernière boîte ouverte, car ce dernier peut contenir des objets. On obtient ainsi la formule suivante :

$$bound = bcount + \frac{capa_{restante} - sumwt}{C}$$

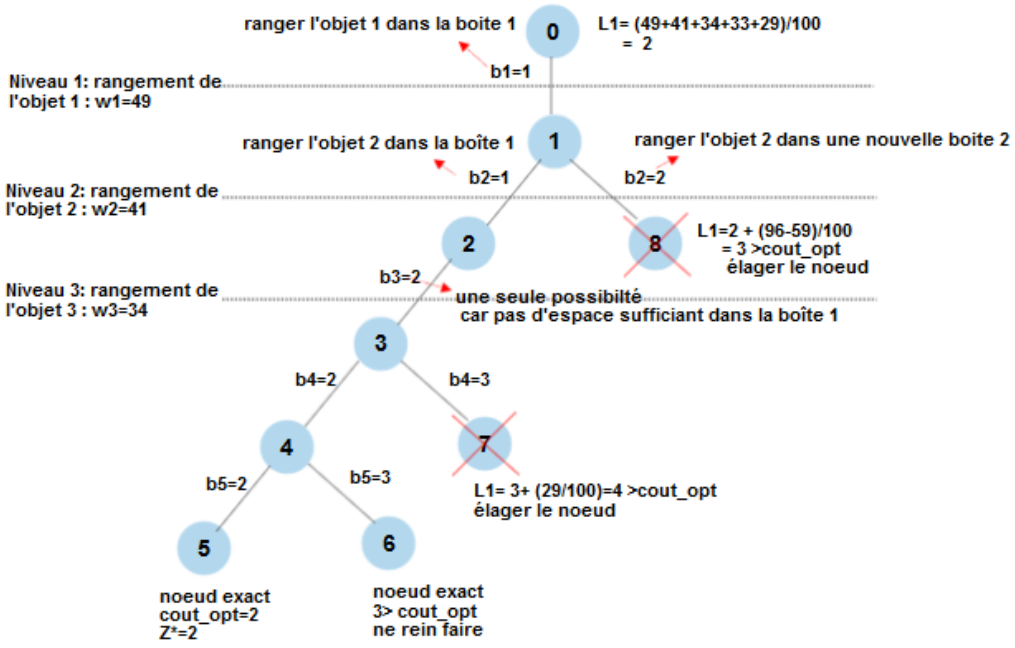
Nombre de boîtes ouvertes

Estimation du nombre de boites à ouvrir encore

exemple 1 : List= 10,50,25,80,70,75,35,70 ; C = 100



bound =((80+70)-20)/100)=2



exemple 2 : n= 5 ; W_j=49,41,34,33,29 ; c=100 on pose : b_j= le numéro de boîte qui contient l'objet j.

3.2 Branch and bound amélioré

Une version améliorée de l'algorithme Branch and Bound présenté ci-dessus. L'amélioration s'est faite en 2 étapes :

1. Utilisation de l'heuristique WFD (Worst Fit Decreasing) pour initialiser la solution optimale.
2. Changement de la borne L1 utilisée par une autre borne plus puissante appelée L2.

Evaluation d'un noeud (Borne L2)

Il a été prouvé que la borne L1 n'est efficace que quand les poids des objets sont petits, c'est à dire qu'on peut mettre plusieurs objets dans la même boîte. Si ce n'est pas le cas, et que les objets ont de grands poids (proches de C) , cette borne n'aura aucun effet et l'algorithme fera une recherche exhaustive. C'est pour cela que la borne L2 à été proposée par

Martello et Toth ,pour remédier à ce problème. on rappelle la formule de la borne L2, qui a été déjà présentée dans l'état de l'art :

rappel Soit α un entier tels que :

$$0 \leq \alpha \leq C/2$$

On definie des classes d'articles suivantes :

$$C_1 = \{a_i, \quad C - \alpha < v_i\}$$

$$C_2 = \{a_i, \quad C/2 < v_i \leq C - \alpha\}$$

$$C_3 = \{a_i, \quad \alpha < v_i \leq C/2\}$$

$BI(I)$ est donnée par la formule suivante :

$$BI(I) = max\{L(\alpha), \quad 0 \leq \alpha \leq C/2\}$$

Avec

$$L(\alpha) = |C_1| + |C_2| + max(0, \lceil \frac{\sum_{j \in C_3} v_j - (|C_2| * C - \sum_{j \in C_2} v_j)]}{C})$$

Explication de la formule : Etant donnée que les objets des classes C_1 et C_2 ont un poids supérieur à $C/2$ chacun d'eux sera placé dans une boîte séparée pour le contenir, donc $|C_1| + |C_2|$boîtes sont utilisées quelque soit la solution. De plus,aucun objet de l'ensemble C_3 ne peut être rangé dans une boîte contenant un objet de C_1 (à cause de la contrainte de capacité). La capacité résiduelle (espace libre) des $|C_2|$ boîtes est de : $C^* = |C_2| * c - \sum_{j \in C_2} w_j$ Donc dans le meilleur des cas, cette capacité résiduelle va être remplie par les objets de C_3, et dans ce cas le nombre de nouvelles boîtes qu'on doit ouvrir est de : $\frac{\sum_{j \in C_3 - C^*} w_j}{c}$ (cette dernière formule utilise le même principe que la borne L1).

3.2.1 Pseudocode

3.3 Recherche exhaustive

Dans cette 3ème solution, on a implémenté une recherche exhaustive, qui consiste à parcourir l'ensemble des nœuds et leurs fils, sans aucun élagage de nœuds. Donc on aura le même algorithme que celui du branch and bound, en supprimant l'étape de l'évaluation du nœud pour décider de son élagage.