

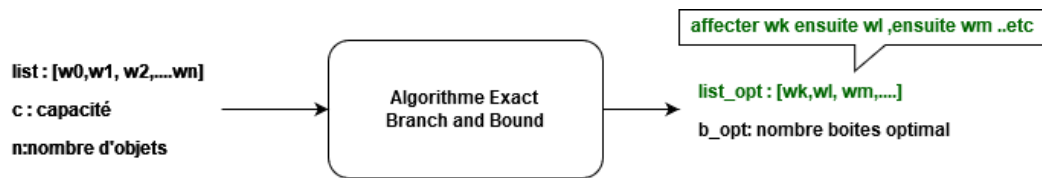
Dans ce chapitre, nous allons présenter la conception détaillée des méthodes exactes sur lesquelles notre choix d'implémentation s'est porté:

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 côté, et sur des instances générées par notre propre générateur d'instances d'un autre côté.

## 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 boîte 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)



## 1.1 Pseudo-Code

Soient:

- $n$  : le nombre d'articles
- $list[0 \dots n-1]$  : la liste des articles en entrées
- $opt\_list$ : la list ordonnée fournissant la solution optimale en sortie
- $opt\_cost$ : le nombre de boîtes optimal
- $C$ :la capacité maximale d'une boîte.

L'algorithme proposé est une fonction récursive `packBins` ayant comme paramètres:

- $k$ :l'ordre de l'élément à être ranger ( le niveau dans l'arbre).
- $sumwt$ :la somme des poids des éléments restants à être rangés
- $bcount$ :la somme cumulée des boîtes déjà utilisées ( depuis la racine jusqu'à ce nœud)
- $capa\_restante$ :l'espace libre restant dans la boîte ouverte.

---

**Algorithm 1** Branch & Bound

---

```
if  $n = k$  then
  //noeud feuille (n objets rangés)
  if  $bcount < opt\_cost$  then
    //solution exacte obtenue par cette branche est meilleure que celle
    trouvée auparavant
    maj du coût optimal  $opt\_cost = bcount$  .
    sauvegarder la solution liste  $opt\_list = liste$ .
  else
    continuer le parcours (monter d'un niveau dans l'arbre).
  end if
else
  //L'ensemble des articles restants sont dans les positions list[k... n-1].

  //chaque noeud fils i signifie qu'on a rangé le ième article parmi les
  articles restants (ayant la position k+i dans la liste list) à la position k.

  for chaque noeud fils i do
    Mettre l'article list[k+i] dans une boîte en permutant l'article[k+i]
    avec l'article[k] : permuter(k+i,k) .
    Incrémenter le nombre de boîtes utilisées (bcount) si on a ouvert une
    nouvelle boîte.
    Mettre à jour la capacité restante (capa_restante).
    Mettre à jour la somme des volume des articles restants à être ranger
    (Sumwt)
    Calculer l'évaluation du noeud fils courant (borne L1) :  $Bound =$ 
 $bcount + \frac{(sumwt - capa\_restante)}{C}$ 
    Comparer l'évaluation du noeud avec la solution optimale courante :
    if  $bcount \geq opt\_cost$  then
      //solution exacte obtenue par cette branche est meilleure que celle
      trouvée auparavant
      le noeud est éliminé. Dans ce cas on re-permute pour revenir à l'état
      précédant (permute (k+i,k)).
      sauvegarder la solution liste  $opt\_list = liste$ 
    else
      on exploite le noeud courant encore, en faisant un appel récursif à
      la fonction avec la valeur k+1 dans le 1 paramètre, en utilisant les
      nouvelles valeurs des autres paramètres.
    end if
  end for
end if
```

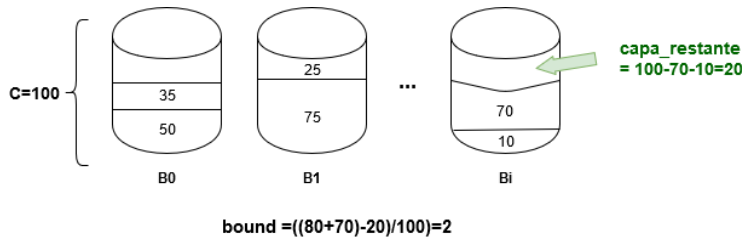
---

## 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 :

$$bound = \underbrace{bcount}_{\text{Nombre de boîtes ouvertes}} + \underbrace{\frac{capa_{restante} - sumwt}{C}}_{\text{Estimation du nombre de boîtes à ouvrir encore}}$$

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

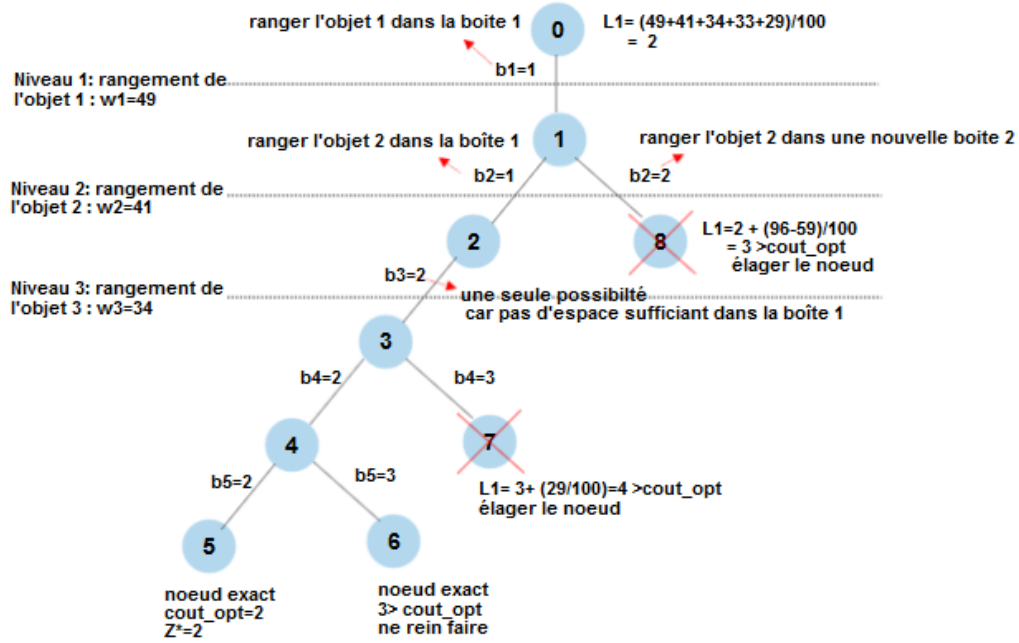


**exemple 2:**  $n = 5$  ;  $W_j = 49, 41, 34, 33, 29$  ;  $c = 100$  on pose :  $b_j =$  le numéro de boîte qui contient l'objet  $j$ .

## 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 a 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  $\alpha$  un entier tels que :

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

On définit 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

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

**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).

## 2.1 Pseudo-Code

---

### Algorithm 2 Branch and bound amélioré

---

Appliquer WFD sur l'instance pour initialiser le coût optimal:  
`cout_opt=WFD(problème)`

Appliquer l'algorithme Branch and Bound sur le problème en utilisant la borne L2

---

## 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.

## 4 La programmation dynamique

**principe** Étant donnée une liste L de N articles à ranger et la capacité d'une boîte C :

### structure de donnée :

- La table de vérité m : une matrice de (C+1) colonnes, et N lignes, tel que  $m[i][j]$  désigne si l'article i peut être rangé dans une capacité j.

### Étapes :

1. Remplir la table de vérité correspondante au problème actuel (L,C), puis ouvrir une nouvelle boîte
2. En parcourant la table de vérité, choisir les articles à mettre dans cette nouvelle boîte.
3. Ranger les articles choisis dans la boîte, et mettre à jour la liste L (retirer ces articles de L).
4. Répéter le processus jusqu'à avoir rangé tous les articles (N=0)

### 4.1 Pseudo-Code

#### 4.1.1 Méthodes utilisées :

la méthode : `get_truth_table(capacité, items):`

#### **paramètres:**

- capacité : la capacité d'une boîte
- items : la liste des articles de poids  $W_j$  à ranger

**Rôle :** Création et initialisation de la table de vérité

---

**Algorithm 3** get\_truth\_table(capacité, items)

---

```
Initialiser toutes les cases de la table de vérité m à "true"
Parcourir les articles un à un //les lignes de la matrice m
for chaque article i do
  for chaque colonne j do
    ///parcourir la capacité une unité par une unité
    if  $i = 0$  then
      //premier article
      if  $j > 0 \text{ et } j \neq W_j$  then
        //c'est à dire cette capacité ne pourra pas accueillir l'article
        courant
        Mettre "faux" dans la case
        //revient à remplir la première ligne par "false", sauf la case 1 et
        la case qui correspond au volume de l'article
      end if
    else
      //pour le reste des articles, utiliser la relation suivante sur la table
      if  $j < W_j$  then
         $m[i][j] = m[i-1][j]$  //la valeur de la case en dessus
      else
         $m[i][j] = m[i-1][j] \vee m[i-1][j - (W_i)]$ 
      end if
    end if
  end for
end for
return m
```

---

**\_pick\_items(m)**

**paramètres:**

- m : table de vérité

**Rôle :** Choisir les articles à mettre dans la boîte



---

**Algorithm 4** `_pick_items(m)`

---

$K = (\text{nombre de lignes de } m) - 1$  // indice de la dernière ligne  
Initialiser la liste des indices des articles choisis à la liste vide :  
`picked_items_indices = []`  
**if**  $k \geq 0$  **then**  
     $k = \max(j \mid \text{telquem}[k][j] = \text{true})$  //prendre la plus grande capacité totale  
**end if**  
**while**  $k \geq 0$  **do**  
    //tant qu'il nous reste encore de lignes à visiter  
    **if**  $k = 0$  **et**  $j > 0$  **then**  
        //première ligne  
        ajouter l'article  $k$  à `picked_items_indices`  
    **else**  
        **if**  $m[k-1][j] = \text{false}$  **then**  
            ajouter l'article  $k$  à `picked_items_indices`  
             $j = j - W_k$   
        **end if**  
    **end if**  
     $k = k - 1$  //allez à la ligne de dessus  
**end while**  
**return** `picked_items_indices`

---

la méthode : `_move_items_to_bin(list_of_items_indices, bin_index)`

**paramètres:**

- `list_of_items_indices` : liste des indices des articles de poids  $W_j$
- `bin_index` : le numéro de la boîte de destination

**Rôle :** Ranger les articles dans la boîte

---

**Algorithm 5** `_move_items_to_bin(list_of_items_indices, bin_index)`

---

**for** chaque article à indice dans `list_of_items_indices` **do**  
    Ranger l'article dans la boîte ayant l'indice `bin_index`  
**end for**

---

#### 4.1.2 Méthode principale :

**La méthode:** `Pack_items()`

**Rôle :** ranger les articles dans un nombre min de boîte , en retournant la solution optimale et son coût ( nombre de boîtes utilisées)

---

**Algorithm 6** `_move_items_to_bin(list_of_items_indices, bin_index)`

---

```
while il reste encore des articles à ranger dans la liste L do
  Construire la table de vérité m : m=get_truth_table(capacité, items)
  bin_index = Ouvrir une nouvelle boîte
  Ajouter la nouvelle boîte à la liste des boîtes
  picked_items = _pick_items(m) // choix des avrticles à ranger dans la
  boîte i
  _move_items_to_bin(picked_items, bin_index) // ranger les articles dans
  la boîte i
  Mettre à jour la liste L, en retirant les articles rangés
end while
```

---

**Exemple :**

Capacité=5;Articles=1,5,2

**première itération** : Construction de la table de vérité:

	C=0	C=1	C=2	C=3	C=4	C=5
w0=1	true	true	false	false	false	false
w1=5	true	true	false	false	false	true
w2=2	true	true	true	true	false	true

- Ouvrir une nouvelle boîte  $B_0$  avec une capacité = 5
- Choisir les articles à mettre dedans : il choisit l'article  $W_1=5$
- Mettre l'article  $W_1$  choisi dans la boîte  $B_0$
- Enlever l'article rangé de la liste des articles à ranger.

**deuxième itération** : Construction de la table de vérité:

	C=0	C=1	C=2	C=3	C=4	C=5
w0 =1	true	true	false	false	false	false
w2 = 2	true	true	true	true	false	false

- Ouvrir une nouvelle boîte  $B_1$  avec une capacité = 5
- Choisir les articles à mettre dedans : il choisit l'article  $W_0=1$  et l'article  $W_2=2$
- Mettre les articles  $W_0$  et  $W_2$  choisis dans la boîte  $B_1$
- Enlever les articles rangés de la liste des articles à ranger.

**troisième itération** :

- Liste vide

ARRÊT DE L'ALGORITHME

**La solution optimale:**

- $B_0$  contiendra l'article  $W_1$  avec un taux d'occupation= $5/5 = 100\%$
- $B_1$  contiendra les articles  $W_0$  et  $W_2$  avec un taux d'occupation= $(1 + 2)/5 = 60\%$